Universidad Carlos III

ESCUELA POLITÉCNICA SUPERIOR

Ingeniería de Telecomunicación

Administración Distribuida de Árboles Multicast a Nivel de Aplicación en Servicios IPTV

Alumno: David Díez Hernández

Universidad Carlos III

ESCUELA POLITÉCNICA SUPERIOR

Ingeniería de Telecomunicación

Administración Distribuida de Árboles Multicast a Nivel de Aplicación en Servicios IPTV

Alumno: David Díez Hernández

Tutor: Jaime José García Reinoso

Resumen

IP multicast es una mecanismo eficiente para distribuir contenido como vídeo a muchos usuarios, pero hoy en día los proveedores de Internet filtran este tipo de tráfico, principalmente debido a problemas de seguridad y tarificación. Para superar este tipo de filtrados, surgió la idea de implementar IP multicast en los propios equipos de usuario, pero a nivel de aplicación. Este tipo de aplicaciones, denominadas en inglés Application Level Multicast (ALM), tienen algunas desventajas, principalmente debido al comportamiento dinámico de los usuarios que se unen y abandonan los árboles de distribución.

Para minimizar el impacto de este comportamiento, en este proyecto se propone realizar un mantenimiento distribuido de ALM que se pueda utilizar para servicios de televisión sobre el protocolo IP (IPTV), en el que algunos nodos del árbol serán los encargados del mantenimiento del mismo. Además, todos los nodos tendrán configurado un *padre adoptivo* que se asignará dinámicamente, y que será al que se tengan que conectar en caso de pérdida de conexión con su actual padre. Esto acelerará el proceso de reconexión de los nodos y minimizará los riesgos de pérdidas en la distribución del contenido para IPTV. Los resultados de este proyecto muestran que se puede conseguir una rápida reconstrucción del árbol a la vez que la carga de trabajo para el mantenimiento y administración del mismo por parte del equipo central es baja, ya que se distribuye entre todos los nodos que componen el sistema.

Palabras Clave ALM (Multicast a Nivel de Aplicación), P2P, IPTV, Video Streaming

A mi familia, a mis amigos y a quien lo entienda.

Cuando va todo al revés,
el cielo sigue en su sitio,
lo que pasa es que no lo ves.
Es mejor confiar,
seguir el camino y andar,
Todo se arreglará,
es cuestión de esperar.

Agradecimientos

En primer lugar, quería darle las gracias a **mis padres y a mis hermanos** *por haberme apoyado en las dificultades de la carrera y por la paciencia que han tenido conmigo todos estos años. Espero que la espera haya merecido la pena.*

Además, hay muchas personas a las que querría agradecer el haber estado apoyándome a lo largo de la carrera. En especial a mis compañeros de Beca, **Joserra y Pinedo***, por haber compartido tantos momentos* always the same*, y sobre todo a* **Lisardo***, que en esta última etapa también ha sido un compañero y un amigo incansable. Y a mis compañeros de carrera,* **David y Rho***, por todos esos momentos de prácticas con prisas y de exámenes sin sentido (Joserra, tú también pero ya te he nombrado antes).*

Y a mis jefes de la beca, **Paco, Jaime e Iván***, que más que jefes, los considero maestros y amigos, y que tanto me han apoyado y guiado en la carrera, en el trabajo, y en todo. Sin ellos no sería posible haber hecho este proyecto. Muy especialmente quería agradecérselo a* **Jaime***, mi tutor del proyecto y la persona con la que más he aprendido desde que entré en la carrera, por la confianza depositada en mí y por su infinita paciencia, y por haber estado ahí tanto en los momentos buenos como en los malos. Muchas gracias Jaime.*

Y en estos agradecimientos no podían faltar mis amigos, compañeros y socios de OGTinc*, que sin ellos estos dos últimos años no habrían sido lo mismo. Gracias por las bromas, por las risas, por las cañas, por los juegos, por las excursiones, por los ánimos, por la paciencia y por ser como sois, que sois los mejores. Os nombraría a la mayoría con detalle y ni siquiera entenderíais por qué (no por vosotros, sino por mí, que no me explico bien), así que os nombro a todos y que cada uno piense por qué. En orden alfabético, como en el gtalk:* **Adal, Aída, Carlitos, Chemi, Dieguete, Elena, Gsus, Lisardo, Miguel, Pablete, Rober, Samu y Virginia***. Espero que este grupo dure muchos años.*

Y por último no me podía olvidar de darle las gracias a **Celeste***, por su amistad, por su confianza, por su apoyo y por su paciencia. Gracias Ce.*

También agradecer a la **Cátedra Telefónica – UC3M en Internet del Futuro para la Productividad** *por la beca que me han concedido durante estos últimos años.*

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En este primer capítulo se explica cuáles son las razones que motivaron la realización de este proyecto y los objetivos que se quieren alcanzar con las ideas planteadas en él. Para ello, se explica brevemente la problemática y las características de los árboles multicast a nivel de aplicación (en inglés Application Level Multicast o ALM) y su utilización para la distribución de canal de televisión sobre IP (IPTV).

1.1. Motivación y objetivos

Hoy en día, para transmitir vídeo en tiempo real utilizando el protocolo IP, el método más eficaz es utilizar *IP multicast* debido a que la información sólo se replica en aquellos puntos (routers) donde haga falta, al contrario que en *unicast*, donde la propia fuente debe replicar el contenido tantas veces como receptores tenga. Sin embargo, el uso de IP multicast en entornos multi-proveedor está restringido debido a la dificultad de tarificar este tipo de tráfico, ya que un paquete en origen puede replicarse varias veces en la red, antes de llegar a sus posibles destinos.

Esto, unido a problemas de seguridad y escalabilidad, han hecho que IP multicast no sea utilizado masivamente en Internet para transmitir vídeo, por ejemplo. Sin embargo, se puede utilizar una técnica en donde se crean árboles de distribución a nivel de aplicación en vez de hacerlo a nivel de red como lo hace IP multicast. A este tipo de técnicas se les llama *Multicast a Nivel de Aplicación* o *Application Layer Multicast (ALM)* en inglés.

En los árboles ALM, los diferentes receptores del contenido pueden actuar a su vez como emisores para otros equipos si disponen de los recursos necesarios, creando así un

árbol de distribución, cuya raíz será el equipo generador de los contenidos (un Media Server, por ejemplo). El árbol tendrá también nodos *hoja* que recibirán el contenido pero no transmitirán dicho contenido a otros nodos y, quizás, nodos *interiores* que recibirán el contenido de otros nodos que a su vez transmitirán el contenido a uno o varios nodos receptores. Se dice que si un nodo N_i envía datos a otro N_j, el primero es *padre* del segundo (y, lógicamente, el segundo será *hijo* del primero).

El principal problema al usar cualquier técnica de ALM reside en el comportamiento dinámico de los usuarios. Si un nodo padre de uno o varios nodos, decide cambiar de canal (o salir del sistema) el sistema deberá reconstruir el árbol recolocando a los hijos que deja *huérfanos*. Esto puede provocar un corte en la reproducción del vídeo en los hijos que han quedado huérfanos, dependiendo del tiempo que se necesite para realizar la reconstrucción del árbol. Por lo tanto, el tiempo de reconstrucción es un parámetro que se debe intentar minimizar.

El tiempo de reconstrucción depende de varios factores y de la técnica utilizada para realizar la reconstrucción. En las *salidas ordenadas*, los nodos envían un mensaje a un nodo central que se encarga a su vez de reconstruir el árbol si fuese necesario. El nodo central conoce la estructura completa del árbol y los recursos disponibles de cada uno de los nodos, por lo que puede decidir cómo recolocar a los nodos huérfanos, balanceando la estructura del árbol resultante. Una vez aplicado el algoritmo de reconstrucción, el nodo central envía la información a cada uno de los nodos huérfanos para asignarles nuevos padres.

Uno de los mayores inconvenientes de este mecanismo es que un nodo central debe tener la información completa de todos los árboles de distribución para que el algoritmo proporcione un resultado óptimo en cuanto a la distribución de carga de los nodos. Además, el tiempo de reconstrucción del árbol depende del tiempo de ejecución de dicho algoritmo y del tiempo de transmisión y propagación hasta recibir la información con el nuevo padre al que se deben conectar.

Como se puede ver, la construcción de este tipo de arquitecturas no está exenta de problemas a pesar de las ventajas que conlleva. En este proyecto se quiere llevar este

sistema un paso más adelante utilizando un mecanismo novedoso basado en la administración del árbol por niveles. La idea es descentralizar la administración, cediendo parte del control del árbol a algunos nodos que lo componen (un elemento de control en cada nivel), y conseguir así que se reconfigure más rápidamente. Esto será posible gracias a que cada elemento encargado de controlar el árbol tendrá menos carga de trabajo en cuanto a tráfico e información de configuración a almacenar. Además, a cada nodo se le asignará dinámicamente un padre adoptivo que le permitirá reconectarse más rápidamente en caso de fallo de su padre actual. Esta técnica mejorará la eficiencia del sistema y la experiencia del usuario, ya que para la retransmisión de vídeo en directo es muy importante que los nodos estén recibiendo el flujo de datos continuamente.

Por tanto, los objetivos serán:

- Diseñar la arquitectura del árbol bajo las premisas mencionadas anteriormente (administración distribuida por niveles y reconexión rápida gracias a la existencia de padre adoptivo).
- Diseñar un protocolo para la comunicación entre los nodos que permita dicho funcionamiento.
- Comprobar la estabilidad del árbol.
- Demostrar que el sistema se comporta de la manera esperada, siendo apto para la utilidad que se le quiere dar, que será la de transmisión de vídeo en directo (IPTV).

Otro objetivo de este proyecto es que sirva para seguir investigando en las posibilidades de este tipo de tecnología, si es que se considera viable. Es importante resaltar que este proyecto lleva una carga de diseño muy alta, siendo en su mayoría ideas originales planteadas y desarrolladas entre tutor y alumno, por lo que es de entender que el abanico de posibilidades que ofrece o que puede llegar a ofrecer es muy amplio, dando lugar a multitud de análisis posteriores, pero muchos de ellos probablemente no abordados en este documento, y que servirán para futuras líneas de investigación a raíz de los resultados obtenidos.

En el siguiente apartado se ahondará en el concepto de ALM y las características de diseño más importantes.

1.2. ALM

El concepto de ALM consiste simplemente en la implementación de la funcionalidad que realizan las redes multicast IP, pero a nivel de aplicación en vez de a nivel de red. En una red multicast IP, los nodos que la implementan se encargan de organizar árboles de distribución que evitan que se produzcan múltiples copias de los paquetes para un mismo enlace y para un mismo conjunto de destinatarios. En el caso de los ALM, los nodos simulan la red multicast enviando los paquetes de manera unicast, pero creando estructuras parecidas a las de una red multicast. Esto quiere decir que los nodos forman una topología lógica multicast, pero la transmisión a nivel físico entre ellos se realiza por unicast. En la Fig. 1.1 se puede ver la diferencia en la construcción de ambas arquitecturas.

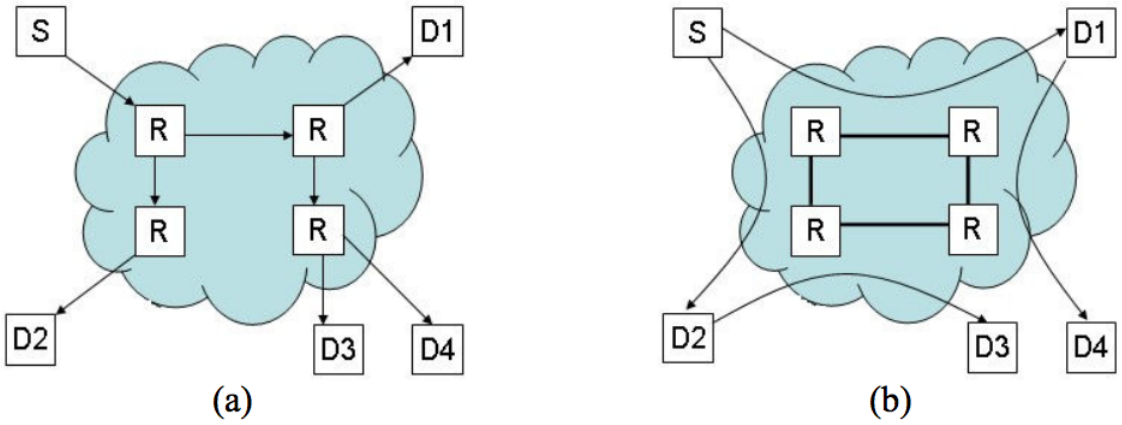

Figura 1.1: (a) Escenario IP multicast (b) Multicast a Nivel de Aplicación

Mientras que en (a) la fuente (S) envía un único paquete, en el ALM (b) tiene que enviar dos, uno para cada uno de los siguientes destinatarios.

- A nivel de red, este tipo de construcciones tiene la ventaja de que son capaces de atravesar Internet, ya que en muchos casos los paquetes multicast se filtran en los routers y no llegan a su destino. Pero en cambio, suelen ser menos eficientes en cuanto a la posible generación de árboles, que da como resultado que se produzcan varias copias del mismo paquete para un mismo enlace. Además, suelen sufrir mayores retardos que con multicast por este motivo.

- A nivel de aplicación, la administración y actualización del algoritmo es más fácil y se puede adaptar a cualquier aplicación. Además, en un ALM, los participantes de la red son los que se encargan del mantenimiento de la red y el envío de la información, abstrayendo de esa tarea a cualquier otro nodo intermedio y a los routers que haya en la red.

A cambio de todo esto, los nodos que componen una arquitectura ALM deben ser más complejos en cuanto a implementación, ya que necesitan manejar más información y no sólo limitarse a recibir contenido, como suele pasar en multicast. Otro problema que tienen es que, como a priori no conocen la topología de la red en la que se mueven, son menos eficientes en cuanto a tiempo de latencia de los paquetes comparado con multicast, ya que darán más saltos.

1.2.1. Decisiones de diseño de ALM

Existen diferentes parámetros que hay que tener en cuenta a la hora de diseñar un protocolo ALM, y éstos estarán principalmente relacionados con la utilidad que se le va a dar y las características de la red en la que funciona. A continuación se describen algunos de estos parámetros.

- **Utilidad de la Aplicación.** Es muy importante definir para qué se va a usar la aplicación, ya que esto determinará el número y tipo de usuarios que deberá soportar el sistema, así como la manera de enviar los datos y cómo se van a optimizar. Existen diferentes tipos de utilidades principales:
  - **Audio/vídeo streaming:** normalmente implica a una única fuente y a un número relativamente grande de usuarios receptores. De este tipo suelen ser las retransmisiones de vídeo en directo. Los parámetros más importantes a considerar deberían ser el ancho de banda y la latencia.
  - **Audio/vídeo conferencia:** suele implicar a un número bajo o medio de usuarios que intercambian vídeo y/o audio entre sí. En este caso suele haber varias fuentes emisoras y varias receptoras y los parámetros más importantes a considerar son los mismos que los del caso anterior.
  - **Servicios Multicast genéricos:** son los que se usan para cualquier tipo de aplicación. Suelen utilizar métricas específicas.

- **Difusión de datos y transferencia de ficheros:** en este caso lo más importante es el ancho de banda y la fiabilidad en la transmisión.

Dependiendo de todas estas aplicaciones y de los parámetros a considerar, será más conveniente utilizar un tipo de ALM u otro.

- **Despliegue de la red.** Un factor determinante a la hora de diseñar un protocolo para ALM es el nivel al que se va a desplegar. Puede ser a nivel de infraestructura o a nivel de extremo o de usuario final.

  - En el caso del nivel de infraestructura, también conocido como basado en proxy, se necesitan servidores/proxys dedicados que organizan la capa de red para proporcionar un servicio multicast transparente al usuario final. Este tipo de despliegue reduce la complejidad de la aplicación a desarrollar, ya que se basan en multicast, pero son menos adaptables y menos optimizables para aplicaciones específicas (dependen de las utilidades de multicast).
  - En el caso de los protocolos ALM a nivel de extremo, se asume un comportamiento como servicio unicast y son los propios usuarios del sistema los que proporcionan la funcionalidad multicast. Tienen la ventaja de ser más flexibles y adaptables, pero no escalan tan bien en cuanto a ancho de banda y otras prestaciones.

  En la Fig. 1.2 se puede ver la diferencia entre ambos tipos de despliegue. En (a) todo el tráfico pasa por los proxies que se encargan de distribuir el contenido utilizando multicast, mientras que en (b) son los nodos lo que envía directamente la información.

- **Mantenimiento del grupo.** Hay varias labores de mantenimiento de los ALM que hay que tener en cuenta a la hora de elegir uno u otro. Es importante determinar cómo se van a unir los usuarios a la sesión o cómo la van a abandonar. Además, es importante decidir si la gestión del servicio se va a hacer de manera centralizada o distribuída. En el primer caso existirá algún elemento que tendrá una visión global de la red y se encargará de administrarla en su mayor parte. En el caso de la gestión distribuida, existirán varios gestores dependiendo de la arquitectura usada. En el siguiente capítulo se verán las diferentes maneras de asociarse los nodos entre sí.

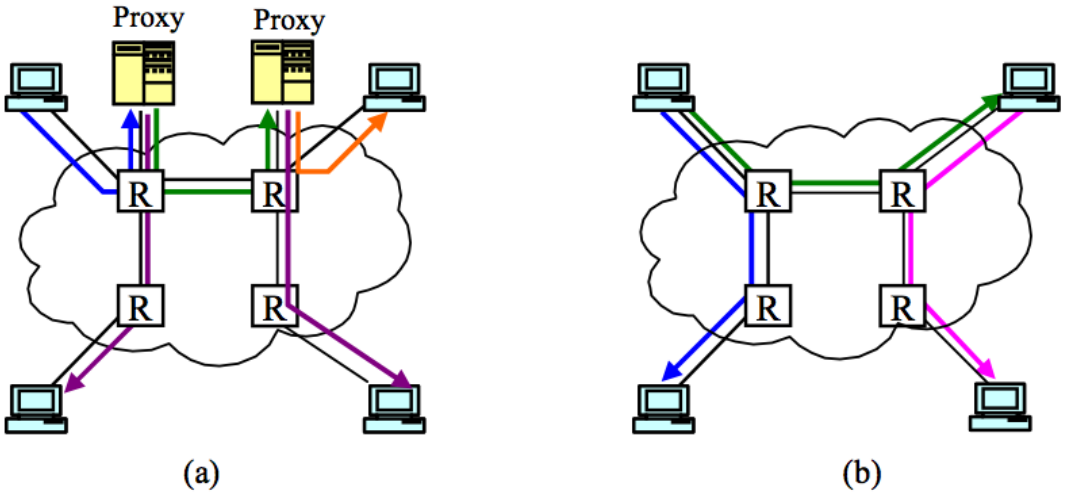

Figura 1.2: (a) Despliegue de ALM basado en proxy (b) Despliegue de ALM extremo a extremo

- **Mecanismo de encaminamiento.** Define cómo se van a conectar los participantes entre sí. Una vez que se han decidido en el diseño los parámetros anteriores, es hora de decidir la construcción del árbol mediante un mecanismo de encaminamiento. Éste mecanismo dependerá de algún parámetro que se quiera minimizar o maximizar, como por ejemplo el retardo entre determinados nodos o la cantidad de tráfico de control generado. Los mecanismos más usados normalmente son:

  - **Camino más corto.** Este método proporciona un camino a través de la red que minimice el número de saltos de los paquetes y consecuentemente, el retardo que se produce. Es relativamente complejo de usar, ya que requiere información del estado de la red.

  - **Estructura en forma de cluster.** Los nodos se agrupan en función de algún parámetro y se forma una estructura jerárquica con ellos, en la que los nodos guardan información de sólo unos pocos nodos de los que dependen. De esta manera se reduce la carga de control.

  - **Estructura P2P.** Los nodos se encaminan unos a través de otros independientemente de la red sobre la que discurren, ya que forman una estructura independiente. La información se va pasando de unos nodos a otros a través de nodos intermedios que forman el árbol. Es el método más usado para la creación de estructuras ALM.

Una vez vistas las características más importantes de los ALM, se puede pasar a ver el resto de documentación incluida.

La estructura del proyecto estará organizada de la siguiente manera. En primer lugar, en el capítulo 2 se hará un repaso del estado del arte, analizando los principales métodos para distribuir vídeo por la red. En el mismo capítulo se verán algunos ejemplos ya implementados. A continuación, en el capítulo 3 se detallará el diseño del sistema ALM propuesto. Posteriormente, en el capítulo 4 se describirá la implementación utilizada para validar el diseño y por último se mostrarán los resultados obtenidos. Finalmente, se ha realizado una planificación del trabajo realizado así como el correspondiente presupuesto (anexo A).

Capítulo 2

Estado del arte

En este capítulo se hace un repaso a los métodos de distribución de vídeo centrándose en las soluciones ALM, sus variantes y algunos ejemplos ya implementados.

2.1. Visión general

Para transmitir vídeo a través de la red tradicionalmente se ha usado una arquitectura cliente-servidor, en la que cada usuario se conecta a un servidor que es la fuente del vídeo. Una modificación sobre esta arquitectura es la denominada CDN (*Content Delivery Network*) en la que la fuente de vídeo se distribuye por un conjunto de servidores situados estratégicamente en la red. De esta manera, el cliente se conecta al servidor de contenido más cercano que tenga, distribuyendo la carga del servidor central. El problema de las arquitecturas cliente-servidor para la distribución de vídeo es la escalabilidad, ya que conforme el número de usuarios aumenta, se necesitan más servidores para balancear la carga. Pero desde hace relativamente poco tiempo se han empezado a utilizar redes P2P para solucionar los problemas que presentan las arquitecturas tradicionales para la distribución de vídeo. En este tipo de redes, los usuarios actúan como clientes y como servidores.

Existen dos tipos de servicios de vídeo para los que se pueden utilizar estas redes. El primero es el de vídeo bajo demanda, en el que el usuario elige reproducir un contenido para visualizarlo cuando quiera. Un ejemplo de este tipo de servicios sería *youtube*[1]. El segundo, que es el que nos ocupa en este documento, es el de vídeo en directo. En este tipo de servicio, todos los usuarios desean recibir un vídeo de una fuente determinada

[1] http://www.youtube.com

que está retransmitiendo en directo. Es decir, los usuarios están sincronizados en la reproducción.

Para la opción de reproducir vídeo en directo, existen actualmente dos soluciones basadas en arquitecturas P2P: las estructuras basadas en árbol (*tree-based*) y las basadas en malla (*mesh-based*).

2.1.1. Estructuras basadas en árbol

Como se ha dicho anteriormente, las distribuciones basadas en árbol pretenden emular los árboles multicast ideados para la distribución de vídeo y audio. Pero dado que el soporte de IP multicast se encuentra muy restringido por parte de los proveedores, se ha intentado emular el comportamiento de éstas mediante la utilización de los ALM.

Podemos diferenciar dos tipos de estructuras basadas en árbol. Las redes de un sólo árbol y las redes de múltiples árboles.

- En las redes de un sólo árbol existe una fuente de vídeo que envía al resto de usuarios ([CRSZ02, BBK02]). Pero a diferencia de la arquitectura cliente-servidor, los usuarios estarán organizados en forma de árbol de manera que cada usuario recibe el vídeo de un único nodo padre y es capaz de reenviarlo a uno o más nodos hijo, dependiendo de los recursos que tenga disponibles. En este tipo de arquitectura reduce la carga que tenía el servidor, ya que el resto de usuarios que forman la red también actúan como retransmisores del vídeo. Existen principalmente tres aspectos que hay que tener en cuenta a la hora de diseñar un sistema de este tipo.

  - El primero es que cada nivel añade un retardo a la retransmisión del vídeo. Por lo tanto un criterio de diseño es mantener el número de niveles de un árbol bajo un cierto límite que dependerá del número de nodos. Para ello cada usuario debe intentar atender al mayor número de hijos posible. Pero esto en la realidad está claramente limitado por el ancho de banda de subida que tiene cada usuario, que hace que no se pueda retransmitir nada más que a un número limitado de hijos, que en general es muy pequeño.
  - Segundo, tampoco es recomendable conectar muchos nodos hijos a un nodo con gran número de recursos, ya que en caso de abandono del padre, se tendrá que reconectar un elevado número de nodos, lo que podría suponer

una reestructuración compleja del árbol. En la Fig. 2.1 se muestra un ejemplo de reestructuración frente al abandono de un nodo. Cuanto mayor es el número de nodos a cargo del nodo que abandona, más complicada se vuelve la reconfiguración.

- El último aspecto importante se refiere al mantenimiento del árbol. Los usuarios pueden entrar y salir del árbol en cualquier momento, y cuando lo abandonan, sus hijos dejan de recibir vídeo. Para que no se produzcan cortes en la reproducción, el árbol tiene que reconstruirse lo antes posible.

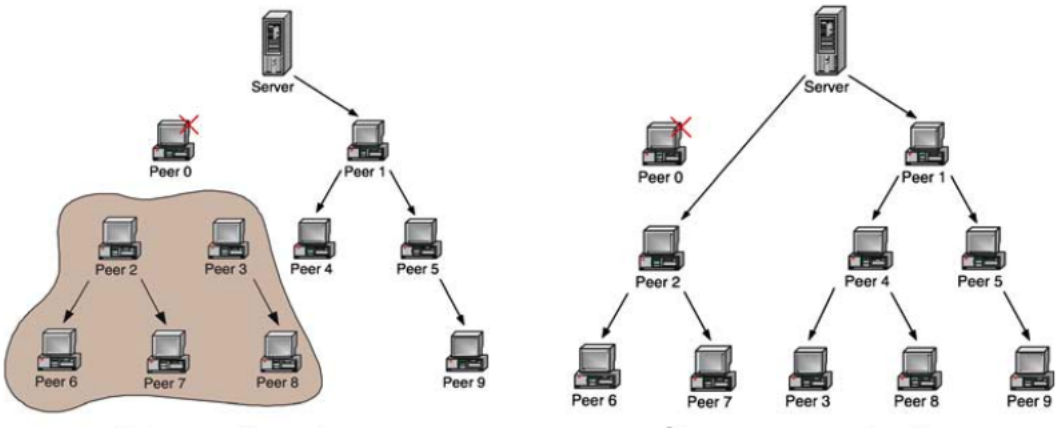

Figura 2.1: (a) Impacto de un abandono (b) Reconfiguración del árbol posterior

En las arquitecturas existentes hasta ahora la construcción y mantenimiento del árbol se realizan de forma centralizada. Es decir, existe un servidor central que controla la posición en la que ha de ubicarse un nuevo usuario y también controla la reconfiguración del árbol en caso de detectar abandonos, ya sea de manera ordenada, por avisos que se produzcan, o desordenada, mediante un soft-state. En cualquier caso, para árboles relativamente grandes, este tipo de sistemas presentan claramente un cuello de botella que puede repercutir en el rendimiento general del árbol, ya que el servidor central tendría que atender a demasiadas peticiones en función del dinamismo de la red que administra. Con lo cual, es posible que el árbol no se pueda reconfigurar suficientemente rápido cuando haya abandonos.

En las redes de múltiples árboles ([CDK+03, KRAV03, NA03]) se intenta aprovechar los recursos de los nodos que están en el extremo final de cada rama del árbol (nodos hoja), ya que son nodos que sólo consumen recursos, pero no se utilizan

para enviar vídeo a nadie más. Una red basada en múltiples árboles pretende solucionar este problema creando varios flujos diferentes del mismo vídeo y haciendo que cada uno se transmita por un árbol diferente. De esta manera, cada nodo puede tener un rol diferente en cada uno de los subárboles, pudiendo ser nodos hoja en un árbol e interiores en otros, cediendo así por lo menos parte de su ancho de banda de subida. En la Fig. 2.2 se puede ver un ejemplo en el que se envían dos flujos y se crea un árbol por cada flujo. En cada árbol, los nodos estarán ubicados en diferentes posiciones de manera que en un árbol sean nodos internos y en el otro nodos hoja.

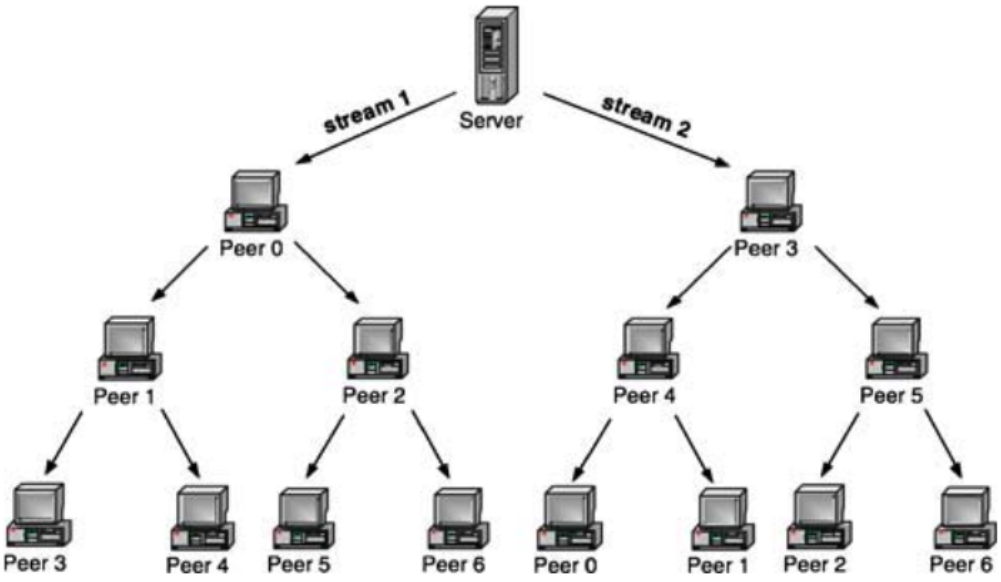

Figura 2.2: Estructura basada en basada en múltiples árboles con dos flujos

2.1.2. Estructuras basadas en malla

Por último, queda por repasar las estructuras basadas en malla. En las estructuras en árbol vistas hasta ahora, cada nodo sólo tiene un padre del que se descarga el vídeo (o un flujo determinado). En caso de abandono de un nodo, los nodos hijos dejan de recibir información hasta que se conectan a otro padre. El mantenimiento del árbol supone un reto en el caso de que se produzcan muchos abandonos, ya que por cada uno ha de reconfigurarse dinámicamente reasignando nuevos padres. Actualmente, hay muchos

sistemas P2P para distribución de vídeo basados en malla. En este tipo sistemas, se establecen relaciones de asociación entre varios nodos a la vez. Estos nodos serán los vecinos, y un nodo podrá subir o bajar contenido de sus vecinos según lo necesite. Si un vecino abandona el sistema, un nodo puede seguir recibiendo información del resto de sus vecinos, de manera que no se pierda la conexión nunca. Esto hace a los sistemas basados en malla muy robustos frente a abandonos.

Existen muchas maneras diferentes para la construcción y mantenimiento de este tipo de topologías, pero básicamente todas consisten en la implementación de un servidor central que suministra una lista de nodos con los que establecer asociación de vecindad, y al cuál se conecta siempre un nodo que quiere conectase al sistema. El nuevo nodo intentará establecer conexión con los nodos de esa lista y a partir de ahí empezará a recibir información en función de la negociación y de los recursos de cada uno.

El principal problema que tienen es que al recibir los datos de varios sitios a la vez, los nodos deberán ordenar y seleccionar el contenido que les llega para conseguir una correcta reproducción, algo muy importante en el caso de transmisión de vídeo.

2.2. Ejemplos de ALM

En esta sección se analizarán algunos métodos ya implementados para distribuir vídeo en la red utilizando técnicas de ALM. Primero se verá un ejemplo de estructura basada en un sólo árbol, denominado ESM. En segundo lugar se analizará un sistema basado en múltiples árboles como es SplitStream.

2.2.1. ESM

ESM (End System Multicast) [CRSZ02] es un sistema que emplea una estructura basada en árbol y cuyo protocolo es distribuido, autoorganizado, y tiene en cuenta el rendimiento. Los árboles se construyen desde la raíz, que es la distribuidora del contenido. El árbol está optimizado principalmente para gestionar el ancho de banda de los nodos, y en segundo lugar el retardo. Se pueden destacar cuatro características de este sistema.

1. **Mantenimiento del grupo:** Cada nodo del ESM almacena información de un pequeño subconjunto de usuarios, como por ejemplo la ruta que tiene que seguir para alcanzarlo. Para unirse al árbol, los nodos tienen que conectarse con la fuente, la cuál les proporcionará una lista aleatoria de usuarios. De esa lista, cada nodo elegirá un nodo que será su padre dentro del árbol. Para aprender información de los

usuarios, los nodos se van pasando periódicamente subconjuntos de sus tablas con información de la última actualización que han recibido de cada miembro. Si algún usuario no se ha actualizado pasado un tiempo, se elimina de la lista

2. **Asociación dinámica:** Cuando un nodo abandona el sistema, continúa mandando datos por un corto período de tiempo, mientras sus hijos buscan nuevo padre según el algoritmo que se describirá más abajo. Esto minimiza los cortes en la red. Los usuarios mandan paquetes de control periódicamente a sus hijos para indicar presencia.

3. **Adaptación en función del rendimiento:** Cada nodo mantiene un control de la tasa de datos que está recibiendo en cada momento. Si el rendimiento disminuye significativamente, se selecciona un nuevo padre. El tiempo de detección para que se produzca esta circunstancia está establecido por defecto en 5 *segundos*, pero depende mucho de la congestión que haya en la red (control de congestión TCP) y se puede adaptar dinámicamente. Hay que tener en cuenta que aunque puede que no esté recibiendo la tasa adecuada, es posible que tampoco tenga mejores alternativas entre sus posibles padres, y entonces el cambio de padre no sería muy fructífero en esas circunstancias.

4. **Selección de padre:** Cuando un nodo A se quiere unir al árbol o se tiene que reconectar, elige un conjunto aleatorio de nodos de los que tiene conocimiento y que no hayan sido probados hasta ese momento, y los prueba. Cada nodo B le responderá con información de su ancho de banda, su retardo, si está saturado o no, y le dirá si es hijo de A. Esta prueba le sirve al nodo también para saber el RTT entre cada nodo. Después de haber esperado 1 segundo para recibir las respuestas de todos los nodos consultados, A escoge de entre los que no son sus hijos al que tenga mejores condiciones, primero evaluando el ancho de banda en función de los recursos que ya tenga, y si esto no está disponible, según el retardo.

Como se ha podido ver, este sistema ESM es relativamente lento en cuanto a tiempos de reconexión, ya que por su naturaleza basada en TCP y las temporizaciones que lleva, necesitaría bufferes de almacenamiento relativamente grandes. El algoritmo intenta compensar esto buscando siempre la mejor alternativa para un nodo y manteniendo durante un tiempo el envío de flujo mientras se desconecta. Pero en el caso de que se produzcan salidas desordenadas, al sistema le sería más difícil recuperarse y no dispondría de esas ventajas.

2.2.2. SplitStream

SplitStream es un sistema en el que el contenido de vídeo a distribuir se divide en k sub-flujos, y cada sub-flujo se envía en su propio árbol multicast. Los usuarios se unen a tántos árboles como sub-flujos quieran recibir. La intención es construir un conjunto de árboles en los que un nodo participante sea nodo interno en un árbol y nodo hoja en el resto. La ventaja de este sistema es que es bastante robusto frente a fallos en los nodos y abandonos inesperados, ya que al comportarse el nodo como interno en un sólo árbol, sólo se perdería uno de los sub-flujos si ese nodo abandona, y afectaría a los hijos que dicho nodo tuviese en el árbol donde es nodo interior. El objetivo principal del diseño es construir un conjunto de árboles que distribuyan la carga del flujo total en función del ancho de banda de los nodos participantes, y además, se pretende que esto se haga de manera descentralizada, escalable y autoorganizable. SplitStream se basa en crear varios árboles de distribución. Cada árbol se crea con un sistema P2P llamado Scribe ([CDKR02]), que a su vez está basado en Pastry ([RD01]), y que crea una capa de superposición que permite localizar nodos que en este caso se utiliza para construir y mantener los árboles. En la Fig. 2.3 se ve un ejemplo de funcionamiento de SplitStream.

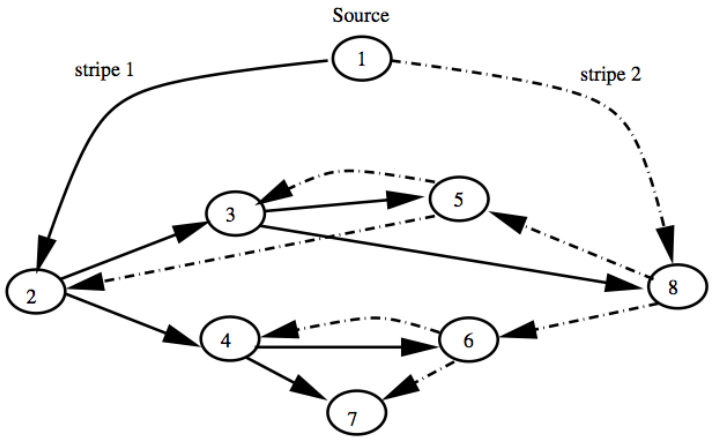

Figura 2.3: Funcionamiento básico de SplitStream con dos sub-flujos que generan dos árboles multicast ([CDK+03])

El objetivo de utilizar la capa de superposición es el de generar un conjunto de árboles disjuntos en los que cada nodo sea nodo interior en tan sólo uno de los subárboles, y en el resto sea nodo hoja. Esto se consigue mediante la utilización de identificadores de nodos que se generan a partir de claves distintas y que permiten que el conjunto de nodos salga

diferente en cada árbol dependiendo de las claves elegidas para cada uno.

En este tipo de sistema, la utilización de varios árboles mejora notablemente el aprovechamiento del ancho de banda de salida de los nodos, pero la manera de buscar nodos con los que conectarse o reconectarse a la red es bastante compleja y requiere tiempos que quizás sean demasiado largos. Aún así, eso puede no ser un problema, ya que al estar conectados a varios árboles, los árboles tienen cierta garantía de seguir recibiendo información en caso de abandonos en alguno de los árboles.

2.3. Ejemplo de Mesh-based

Los sistemas basados en malla son arquitecturas en las que los usuarios intercambian la información del vídeo (o cualquier otro contenido) entre ellos, formando topologías lógicas independientes de la red en la que estén, y además con la característica de que cada nodo puede tener más de una fuente de la que recibir el vídeo, según sus necesidades, y enviar a diferentes nodos, dependiendo de sus recursos. A continuación se va a ver un ejemplo de un sistema de este tipo, como es CoolStream ([LQK+08],[XLKZ07],[ZLL05]).

2.3.1. Coolstream

CoolStream es un sistema que se basa en la selección aleatoria de nodos para asociarse entre ellos y establecer así un intercambio del flujo del vídeo. Además, va montado sobre un estructura P2P centralizada en un nodo bootstrap. CoolStream tiene tres módulos básicos de gestión.

- El primero se encarga de la gestión de los miembros, y mantiene una visión de la red sobre la que se implementa.
- El segundo se encarga de la gestión de usuarios, y es el que establece las conexiones TCP entre los nodos y actualiza la disponibilidad de los bufferes.
- Y el tercero se encarga de la gestión de los flujos, que es el que decide cuáles y de dónde extraer los bufferes para el flujo de vídeo.

Para su distribución y funcionamiento, el flujo de vídeo se divide en varios bloques y a cada uno se le asigna un número de secuencia que representa el orden de reproducción (Fig. 2.4). Una de las ventajas de la compartición de información mediante P2P es la

posibilidad de recibir la información de varios sitios a la vez, requiriendo a cada fuente un trozo diferente. En Coolstream cada nodo se puede suscribir a un conjunto de nodos que son los que le envían la información. Cabe recalcar que en el caso del vídeo este procedimiento no es trivial, ya que está sometido a órdenes temporales y a la obtención de determinadas partes.

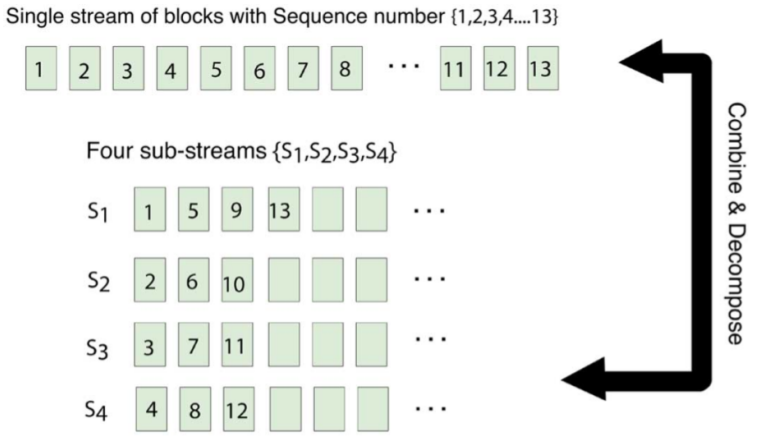

Figura 2.4: Ejemplo de descomposición de flujos en Coolstream ([XLKZ07])

En el sistema Coolstream se habla de una relación de compañerismo entre nodos, en vez de una relación padre-hijo, porque aquí los nodos establecen una conexión TCP en la que pueden pasarse los datos que necesiten en las dos direcciones.

Con el fin de poder reproducir correctamente el vídeo, Coolstream introduce el concepto de *buffer map*, que sirve para representar la disponibilidad de los últimos bloques de diferentes sub-flujos que se quieran reproducir. Esta información se debe intercambiar entre los nodos periódicamente para determinar a qué sub-flujos suscribirse. Por esa razón, la gestión de miembros es muy importante para la construcción y mantenimiento de la red. Además, cada nodo tiene un identificador único y mantiene una caché de miembros activos en el sistema. De esta manera, un nodo utiliza la información de dicha tabla para establecer una conexión TCP con otro nodo. Los elementos que forman el sistema se basan en una única fuente origen, un nodo bootstrap en el que se registran los nodos, y el resto de nodos miembro que son los que intercambian la información de la fuente entre ellos.

El proceso de unión es bastante sencillo. Cuando un nodo quiere unirse a la red, consulta al bootstrap para que le proporcione un conjunto aleatorio de nodos activos. De esta lista, el nodo selecciona un pequeño conjunto de nodos con los que asociarse estableciendo una conexión TCP con ellos. Una vez establecida la asociación, los nodos empiezan a intercambiar información de sus tablas de miembros. Estas tablas se reconfiguran dinámicamente en función de las conexiones y desconexiones y de los recursos de cada miembro. Además, cada nodo se podrá reasociar con otros nodos diferentes dependiendo de sus necesidades.

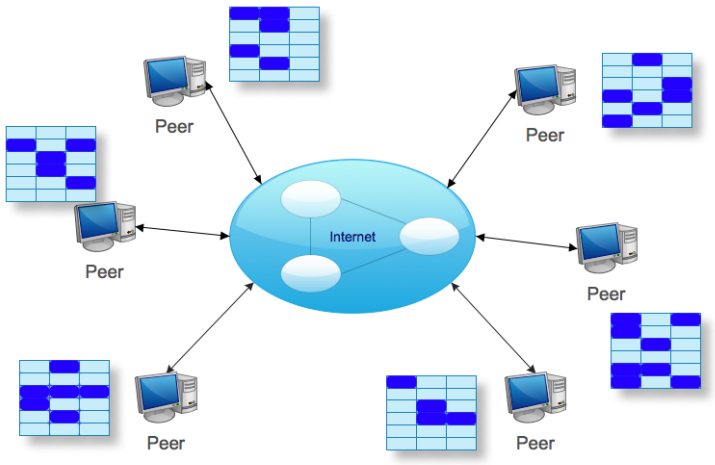

Figura 2.5: Funcionamiento de una red basada en CoolStream

El envío de contenidos se realiza utilizando un esquema *push-pull*, en el que una vez establecida la asociación entre dos usuarios, se intercambian los bloques disponibles de sus *buffer maps* para ver qué bloques necesita cada nodo. En principio el que realiza la petición le dirá al que la recibe que le envíe ciertos bloques disponibles a partir de uno dado, formando una especie de ventana, y el nodo lo empezará a hacer de manera automática según las indicaciones del primero. En la Fig. 2.5 se puede ver un ejemplo del intercambio de los *buffer maps* a través de los usuarios que forman la red.

Los sistemas basados en malla son arquitecturas bastante robustas en cuanto a la posibilidad de estar recibiendo vídeo, ya que cada nodo se asegura siempre de tener a un conjunto suficiente de usuarios que le estén proporcionando sus *buffer maps* y así siempre

tener datos que mostrar. Pero por otro lado, son estructuras relativamente complejas que requieren un tratamiento complejo de los bufferes de recepción, y que además no deja de estar sujeta a discontinuidades en la reproducción, debido a la selección aleatoria de compañeros con los que asociarse, que puede crear incertidumbre sobre los datos a recibir.

Capítulo 3

Descripción de la Propuesta

En este capítulo se tratarán los detalles más importantes de la arquitectura diseñada y el funcionamiento del protocolo desarrollado. En primer lugar se comentarán los objetivos más importantes del diseño. Lo siguiente será describir algunos conceptos básicos sobre la funcionalidad de los elementos que integran el árbol. A continuación se explicarán con detalle los mecanismos que determinan el comportamiento del sistema y finalmente se repasarán los mensajes necesarios para el funcionamiento del protocolo.

3.1. Características del diseño

Como se dijo en la introducción, con este diseño se quieren llevar los sistemas ALM un paso mas allá, utilizando un sistema distribuido que además mejore las prestaciones en cuanto a tiempo de reconexión y tiempo de reestructuración del árbol frente a abandonos. Asimismo, se pretende que la arquitectura sea lo más sencilla posible.

A continuación se describirá la solución planteada para distribuir la carga de construcción y reconstrucción de los árboles multicast a nivel de aplicación que se utilizarán para la transmisión de vídeo IPTV. Se busca una arquitectura distribuida, descentralizada y uniforme:

- **Distribuida:** Tal y como se ha diseñado la red, en forma de árbol, los miembros que la forman tienen una visibilidad fronteriza organizada en niveles.

- **Descentralizada:** El control se establece entre un conjunto de nodos que forman el árbol, en lugar de un único nodo que tenga toda la información del sistema. En este caso, cada elemento de control sólo tendrá información de lo que ocurre en su nivel y en el nivel inmediatamente inferior. Los cambios en una zona de control no

afectan a cambios en otras zonas, distribuyendo por lo tanto la información por los niveles del árbol.

- **Uniforme:** Es una arquitectura uniforme ya que cualquier elemento que la compone es susceptible de asumir el rol de ser elemento de control. No existen nodos dedicados que realicen una función específica, sino que cualquier nodo puede ser padre, hijo o intervenir en el mantenimiento. La única excepción será el nodo Bootstrap server.

Una consecuencia de este tipo de arquitectura es una reducción considerable del número de mensajes en el nodo central y un mejor aprovechamiento del ancho de banda de la red para la distribución de los contenidos, ya que en otro tipo de arquitecturas totalmente centralizadas hay que mantener comunicación con todos los nodos que forman la estructura para informar de cambios. Sin embargo, en este caso, al tratarse la información de control por niveles, los mensajes que hay que enviar cuando se produce un cambio sólo se transmiten entre niveles adyacentes. Además, al producirse los mensajes sólo en el entorno de su nivel, la recopilación de información es más sencilla y rápida, y la cantidad de información que hay que recopilar y almacenar en las tablas de cada nodo también es mucho menor que en otras arquitecturas P2P.

El otro aspecto novedoso de este sistema es la inclusión en cada nodo de la figura del padre adoptivo que será al que se conectará el nodo en caso de desconexión de su padre actual. Esto permitirá que las reconexiones sean más rápidas, ya que los nodos no tendrán que ejecutar ningún algoritmo de búsqueda de padre cuando pierden el suyo. Este algoritmo se ejecuta previamente por el nodo que controla el nivel, que es el que tiene toda la información y puede decidir mejor cómo se realiza la reconexión.

3.2. Elementos de la red

Con el fin de introducir algunos conceptos y definiciones que son necesarias para desarrollar la propuesta, en esta sección se va a describir la funcionalidad de algunos elementos importantes de la arquitectura propuesta y que se representan en la Fig. 3.1.

- **Nodo.** Un Nodo es un equipo de usuario que quiere conectarse a un canal para visualizarlo, y para ello tiene que entrar a formar parte del árbol de distribución de ese canal. Cualquier Nodo, cuando se conecta a un árbol, recibirá el vídeo de un único Nodo, que será su padre, y es susceptible de retransmitir ese vídeo a otros nodos, que serán sus hijos, cuyo número dependerá de los recursos que tenga disponibles. Además, el Nodo sólo enviará y recibirá vídeo relacionado con ese canal. Es decir, que mientras está en un árbol, sólo utilizará contenido manejado por ese árbol.

  Cada Nodo tienen un identificador único que lo identifica unívocamente dentro de la red. Un dato importante que deben tener todos los nodos es quién va a ser su siguiente padre o padre adoptivo, y esta información se la comunica siempre el Big-Parent de su nivel superior. Así, cuando se pierda la comunicación de un Nodo con su padre actual (que es el que le está enviando el vídeo), el Nodo podrá reconectarse directamente a otro padre[1].

  En la Tabla 3.1 se muestra la información que contiene un Nodo de este tipo.

	Padre	Siguiente Padre	Big-Parent	Big-Parent Nivel Superior	Hijo 1	...	Hijo n
ID							

Tabla 3.1: Información contenida en un Nodo

  El ID en realidad estará formado por un conjunto de datos (aparte del propio identificador), como son la IP, el puerto, y el tipo de Nodo al que se referencia.

- **Big-Parent.** Dentro de cada nivel i existe un único nodo denominado Big-Parent que contiene almacenada información sobre el resto de los nodos de ese nivel i, así como de sus recursos. El Big-Parent constituye el punto de entrada para un Nodo que se quiera conectar a un nodo del nivel i, ya que es el Big-Parent el que le

[1] En caso de error de conexión, se intentará una conexión nueva al árbol.

asignará un padre de entre los nodos de su nivel. También se encarga de asignar el *padre adoptivo* a todos los nodos del nivel inferior y de decidir cuál va a ser el Big-Parent de entre ellos. El Big-Parent es un Nodo normal del árbol que recibe y envía vídeo como cualquier otro nodo, sólo que además tiene un papel organizativo dentro de la estructura. Cualquier nodo es candidato a ser Big-Parent cuando entra al árbol y además, cuando un Nodo pasa a ser Big-Parent, no abandona ese rol hasta que abandona el canal. Si abandonase el canal, sería necesario nombrar a otro Nodo para la administración del nivel.

Además de toda la información que contiene un Nodo normal, el Big-Parent debe tener conocimiento del Big-Parent del nivel inferior e incorporar una tabla más (Tabla 3.2) con los Nodos que forman su nivel

Nodos del Nivel	Lista de Hijos (Recursos Usados)	...
ID 1		
...		
ID N		

Tabla 3.2: Información contenida en un Big-Parent

- **Generador.** Es el nodo fuente y emisor del vídeo. Este nodo no puede desaparecer, ya que constituye el origen del vídeo y es la raíz del árbol para ese canal. A efectos de administración del canal, el comportamiento de este nodo es igual que el de un Big-Parent.

- **Bootstrap.** Es el nodo en el que están registrados los canales. Cada canal cuenta con un conjunto de nodos Big-Parents, con los que iniciar la comunicación y que se guardan en el Bootstrap de forma dinámica. Cualquier nodo que quiera conectarse a un canal debe comunicarse antes con el Bootstrap. La manera en que el Bootstrap indica a un nodo con quién debe conectarse se explicará más adelante.

  En la Tabla 3.3 se muestra la información que contiene un nodo Bootstrap.

  En este caso, el Bootstrap, en la meta-información contenida en el ID, también sabrá el estado en el que se encuentra el Big-Parent. El motivo de esto se verá también más adelante.

Ahora que se sabe de qué elementos consta la red, se puede pasar a detallar cómo es la arquitectura.

Canal	Lista de Big-Parents (ID+Estado)	...
1	BPs canal 1	
...		
M	BPs canal M	

Tabla 3.3: Información contenida en un Bootstrap

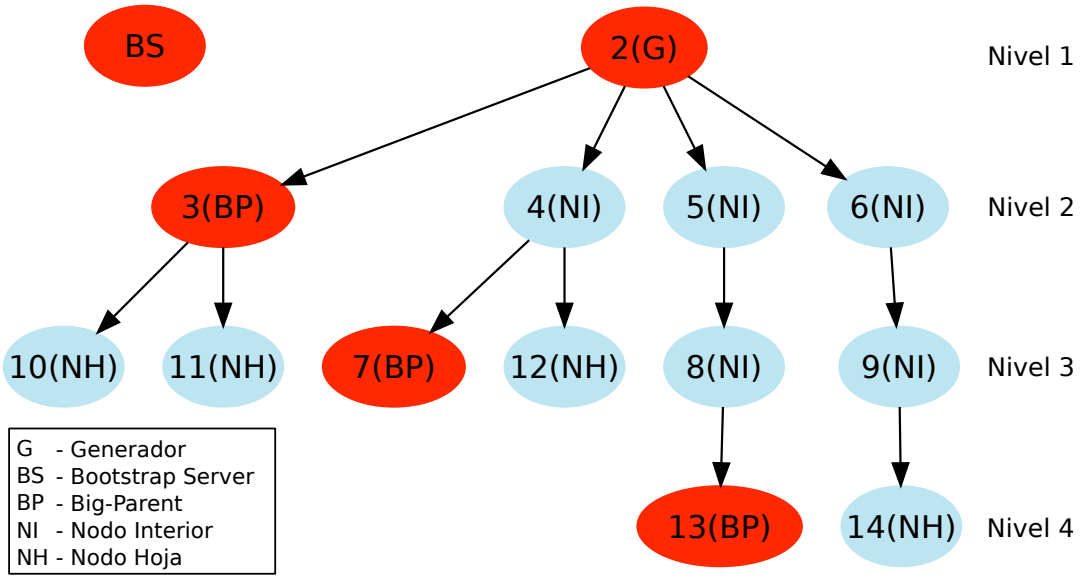

Figura 3.1: Tipos de Nodos

3.3. Estructura de los árboles

La estructura de la red (Fig. 3.1) está formada por un nodo Generador que es el origen del vídeo y raíz del árbol, y el resto de nodos se organizan por niveles. Cada Nodo puede tener un sólo padre y cero, uno o varios hijos, dependiendo de las circunstancias y de los recursos disponibles en dicho Nodo. La dimensión vertical del árbol está organizada en niveles, entendiendo que el nivel 2 estará formado por los nodos que tienen como padre directamente al nodo Generador (Nivel 1), y el siguiente nivel será el formado por los nodos que tengan como padres a los nodos del nivel anterior y así sucesivamente. Cada nivel tendrá su propio nodo Big-Parent, que se se encargará de administrar los recursos de los nodos de su nivel y también de asignar siguiente padre y Big-Parent entre los nodos del nivel inferior. A nivel de nomenclatura, el nodo Generador será el nivel 1, y los siguientes niveles serán 2, 3, 4, etc. a medida que vayan apareciendo en el árbol. Se

dirá que el nivel superior a un nodo que está en el nivel i será el nivel de su padre $(i-1)$, y el nivel inferior el de sus hijos $(i+1)$.

3.3.1. Creación de un Árbol

Inicialmente, el árbol sólo cuenta con el Nodo Generador, que es la fuente emisora del vídeo para ese canal. Cuando un Nodo quiere conectarse a un canal, debe preguntar al Bootstrap server, el cual responderá con la dirección IP y puerto de un Big-Parent de ese árbol (al principio, como sólo está el Generador, ese será el que le asigne el Bootstrap). El siguiente paso es ponerse en contacto con el Big-Parent que le asignará un nodo padre o redigirá la petición a otro Big-Parent. La manera de elegir esto se tratará más adelante, en el apartado 3.5.1. Una vez que el Nodo se ha conectado en el nivel i, el Big-Parent del nivel superior $(i-1)$ puede convertir al nuevo Nodo en Big-Parent de su nivel (i), si éste es el primer nodo de dicho nivel.

3.4. Mensajes

Para el correcto funcionamiento del sistema, se ha diseñado específicamente un protocolo de comunicaciones entre Nodos que se encarga de administrar el árbol multicast. En esta sección se describen los tipos de mensajes que intervienen en el protocolo. Se distinguen tres tipos de mensajes, que son los mensajes de conexión, los mensajes de abandono, y los mensajes de actualización.

3.4.1. Mensajes de conexión

Estos mensajes son los encargados de establecer la conexión de un posible usuario con el árbol de distribución.

- **Join Request**

  Lo envía un Nodo que quiere conectarse al árbol. Contiene información de la identidad de ese nodo, mediante el ID, que consta del identificador único, la IP y el puerto. El mensaje es utilizado para obtener información de conexión al Bootstrap, a un Big-Parent (o al Generador) o a un Nodo normal. Si el mensaje va dirigido al Bootstrap, además se le indicará el canal al que quiere conectarse. En la Fig. 3.2 se representa el origen y los posibles destinatarios de este mensaje.

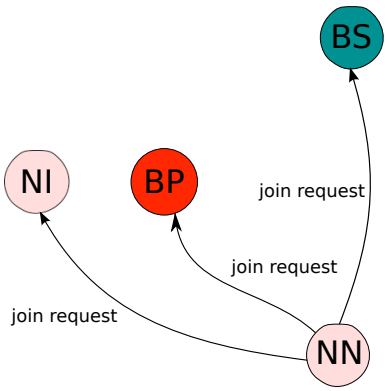

Figura 3.2: Mensaje Join Request

- **Join Reconnect**

  Es parecido al anterior, pero en este caso la conexión es directa al destino del mensaje. Lo envía un Nodo a su padre adoptivo cuando ha perdido la conexión con su padre actual. En el mensaje se indica quién era el padre del nodo hasta ese momento, y de qué tipo era. En la Fig. 3.3 se puede ver un ejemplo en el que tras el abandono de su padre, el hijo se reconecta.

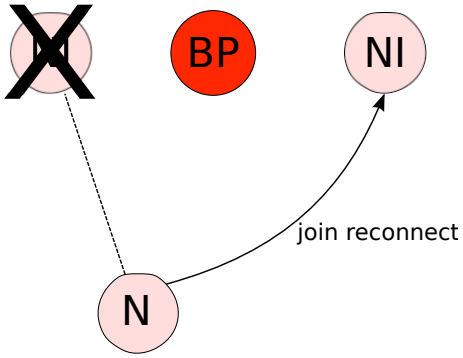

Figura 3.3: Mensaje Join Reconnect

- **Join Reply**

  Es el mensaje de respuesta tanto a un Join Request como a un Join Reconnect y en este mensaje se indica a qué Big-Parent preguntar (en el caso de que lo envíe el Bootstrap) o a qué Nodo conectarse (en el caso de que lo envíe el Big-Parent). En la Fig. 3.4 se muestra los posibles orígenes y destinatario del mensaje.

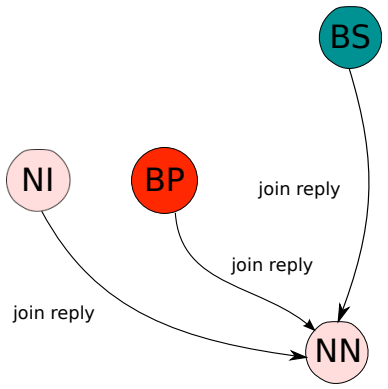

Figura 3.4: Mensaje Join Reply

3.4.2. Mensajes de abandono

- **Leave**

  Para abandonar un canal, un Nodo sólo necesita enviar un mensaje de tipo leave a sus hijos y a su padre, con lo que quedará totalmente liberado del canal. En este mensaje sólo se incluye la información del ID del Nodo. En la Fig. 3.5 se muestra la desconexión de un nodo y los mensajes leave que genera.

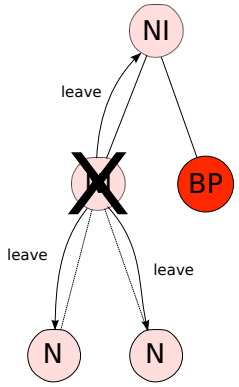

Figura 3.5: Mensaje Leave

3.4.3. Mensajes de actualización

Este tipo de mensajes es el más abundante en el sistema, ya que cuando se produce una conexión o un abandono, es necesario actualizar la información que contienen algunos nodos para reconfigurar el árbol. Hay cuatro tipos de mensajes de actualización, que veremos a continuación.

Updates de nivel

Son mensajes de actualización que se envían entre nodos que están en el mismo nivel. El mensaje se envía siempre desde un Nodo normal hacia el Big-Parent de su nivel. En todos estos mensajes, el Nodo siempre enviará la tabla con todos los recursos ocupados que tenga hasta ese momento, así como su estado y el tiempo que lleven conectados. En la Tabla 3.4 se representa esquematizadamente la información que llevan los mensajes de actualización de este tipo.

Nodo	Lista de Hijos	...
ID	ID + estado + tiempo	

Tabla 3.4: Información contenida en un mensaje Update de nivel

- **Update Join**

  Lo envía un Nodo al Big-Parent de su nivel cuando se ha conectado un Nodo a él y sirve para informar al Big-Parent que tiene un recurso menos disponible. En la Fig. 3.6 se puede ver un update de este tipo.

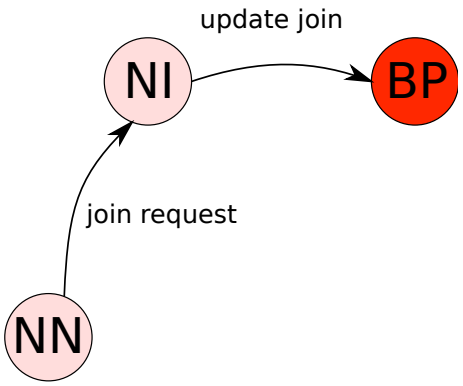

Figura 3.6: Mensaje Update Join

- **Update Leave**

  Igual que el anterior, pero para indicar que ha abandonado un hijo y tiene por tanto un recurso más disponible. En la Fig. 3.7 se muestra un ejemplo de envío de este mensaje.

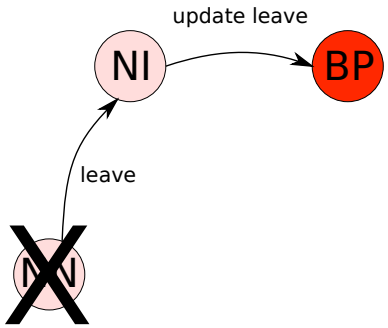

Figura 3.7: Mensaje Update Leave

- **Update Resources**

  Lo envía un nodo al Big-Parent de su nivel para informarle sobre los recursos que tiene disponibles y utilizados. Se produce después de una conexión o tras una actualización del Big-Parent. En la Fig. 3.8 se muestra un ejemplo en el que se envía este mensaje, tras una conexión de un nuevo Nodo al árbol.

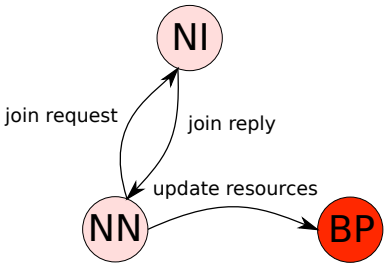

Figura 3.8: Mensaje Update Resources

- **Update Reconnect**

  Lo envía un nodo al Big-Parent de su nivel para indicarle que se ha reconectado a él un Nodo del nivel inferior. El objetivo es indicarle al Big-Parent que el nodo ha cambiado de padre, y que han cambiado los recursos del nodo que envía el mensaje, y asignar al nuevo hijo un nuevo padre adoptivo. En la Fig. 3.9 se ve un ejemplo donde se utiliza este mensaje.

Updates hacia el nivel inferior

Son mensajes de actualización que se envían desde el Big-Parent de un nivel a los Nodos del nivel inferior. Sirven para reconfigurar la información que tengan los Nodos de ese nivel.

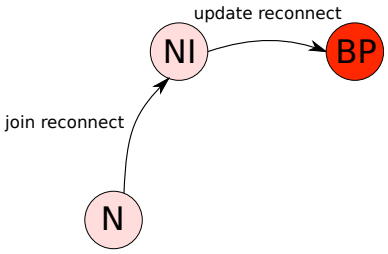

Figura 3.9: Mensaje Update Reconnect

- **Update New BP**

  Lo envía el Big-Parent de un nivel a todos los nodos del nivel inferior (sean sus hijos o no), para comunicarles cuál va a ser el Big-Parent en ese nivel. En la Fig. 3.10 se ve un ejemplo en el que un Bip-Parent, tras enterarse que se ha quedado sin Big-Parent el nivel inferior, envía este mensaje a todos los nodos del nivel inferior con el identificador del nuevo Big-Parent (Nodo 15 en el ejemplo).

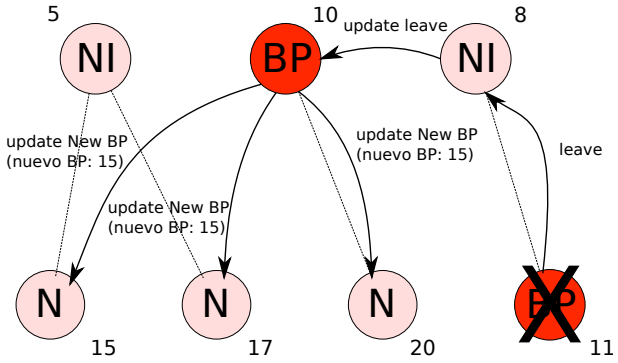

Figura 3.10: Mensaje Update New BP

- **Update New PA**

  Lo envía el Big-Parent de un nivel a un nodo del nivel inferior cuando éste no tenga bien configurado su padre adoptivo, o su Big-Parent. Esto lo sabrá gracias al contenido de las tablas que le lleguen con los mensajes de actualización. En la Fig. 3.11 se ve un ejemplo en el que un Nodo recibe este mensaje tras una reconexión, ya que ha perdido a su padre adoptivo (ahora es su padre actual) y necesita uno nuevo.

- **Update Leavel (Leave+Level)**

  Lo envía el Big-Parent de un nivel al Big-Parent del nivel inferior para indicarle que hay un nodo menos en su nivel. En la Fig. 3.12 hay un ejemplo en el que se envía

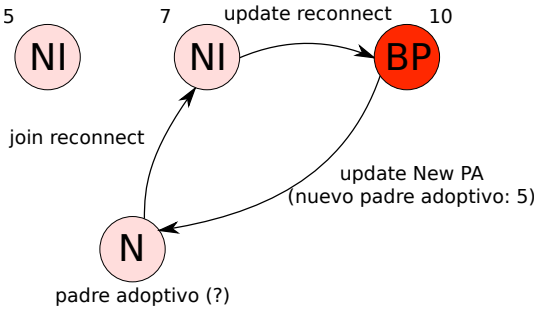

Figura 3.11: Mensaje Update New PA

este mensaje.

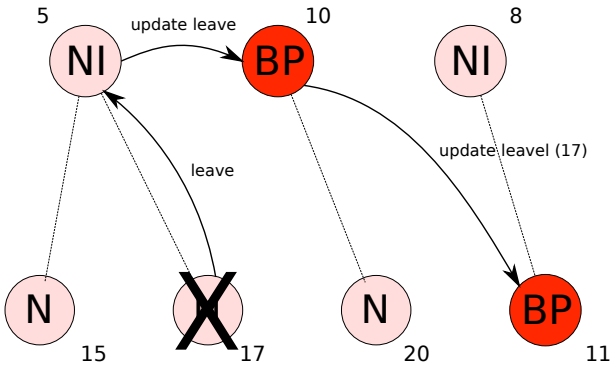

Figura 3.12: Mensaje Update Leavel

Updates hacia el nivel superior

Son mensajes que mandan los nodos de un nivel para actualizar información al Big-Parent del nivel superior.

- **Update ImBP**

  Lo envía el Big-Parent de un nivel i al Big-Parent del nivel superior $i-1$ para indicarle que él es el Big-Parent en su nivel. Se envía cuando el Big-Parent del nivel i se da cuenta de que ha cambiado el Big-Parent del nivel superior, para actualizarlo.

- **Update IneedBP**

  Lo envía el Nodo que ha reconectado a un Nodo cuyo padre anterior era el Big-Parent del nivel, es decir, el Big-Parent del nivel i ha abandonado el canal, y uno o varios de sus hijos se han conectado al Nodo que envía el mensaje al Big-Parent

de nivel superior $i-1$. Va dirigido al Big-Parent del nivel superior, por si este no tuviera constancia de ello. Se envía transcurrido un tiempo que se considera razonable como para que esa información ya estuviera actualizada. En la Fig. 3.13 se reproducen las condiciones en las que se daría.

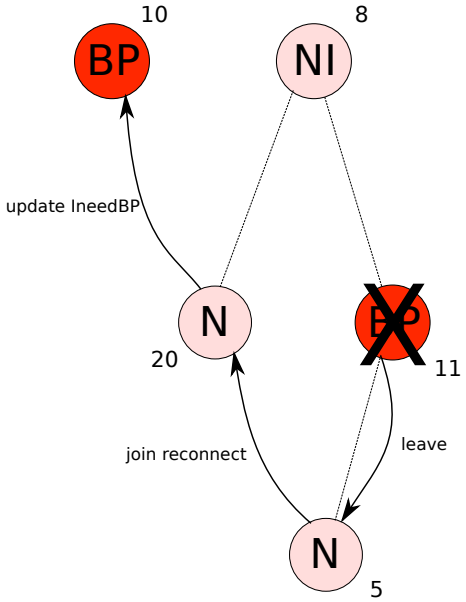

Figura 3.13: Mensaje Update IneedBP

Updates hacia el bootstrap

Estos mensajes los envían los Big-Parents para actualizar la información de estado del Bootstrap. En todos estos mensajes, la información que contendrá será su ID, incluyendo su estado actual.

- **Update Add BP**

  Cuando un nodo se convierte en Big-Parent porque es el primer nodo en su nivel, envía este mensaje al Bootstrap para que lo añada en su tabla de Big-Parents

- **Update Remove BP**

  Lo envía un Big-Parent cuando abandona el canal y no tiene más nodos ni en su nivel i ni en el siguiente $i+1$. El Bootstrap, cuando lo recibe, borra ese Big-Parent de la tabla para no asignarlo más a los Nodos nuevos que se quieran conectar al árbol.

- **Update Leaving BP**

  Lo envía un Big-Parent cuando abandona el canal para indicar al Bootstrap que de momento no hay Big-Parent en ese nivel. El Bootstrap marca el estado de ese nivel para no asignarlo temporalmente.

- **Update Replace BP**

  Lo envía un Nodo cuando sabe que se ha convertido en Big-Parent y avisa al Bootstrap para reemplazar al Big-Parent que hubiera en ese mismo nivel.

- **Update Few Nodes**

  Lo envía un Big-Parent cuando se da cuenta de que tiene pocos Nodos en su nivel, para que el Bootstrap le de prioridad en la asignación de nuevos Nodos al árbol. Esto es necesario para mantener un número adecuado y razonable de nodos en un nivel, en cuanto a disponibilidad de recursos se refiere.

3.4.4. Complejidad del protocolo

Como se ha podido ver, el conjunto de mensajes que conforman el protocolo es relativamente pequeño, estando claramente identificadas las labores que realiza cada mensaje, así como orígenes y destinatarios de los mismos. Cabe destacar también su sencillez, limitándose a enviar información almacenada en tablas y algún otro campo de identificación. Por lo tanto, la codificación de los mensajes será fácil de implementar.

Además, una vez definida la estructura, se puede ver que ofrece muchas posibilidades en cuanto a escalabilidad, pudiéndose ir añadiendo opciones nuevas en los mensajes que ya existen y que ayuden a la reconfiguración del árbol. Pero todo esto no funcionaría sin que los nodos tuvieran la capacidad de abstraer la información contenida en los mensajes y actuar en consecuencia, dependiendo de las circunstancias. De todo eso se hablará en la sección siguiente.

3.5. Mecanismos

En esta sección se explicará el funcionamiento de los mecanismos que permiten a los nodos interaccionar dentro del árbol. En realidad, los mecanismos de adición y eliminación de Nodos son bastante complejos y dependen mucho de la casuística de la situación. Por

ejemplo, no es lo mismo que se vaya un Big-Parent que se vaya un Nodo normal, y como hay que tratar a todos los usuarios del canal por igual para que puedan ver el vídeo sin cortes, es necesario contemplar todos los casos posibles. Para poder abarcar todos ellos se ha provisto de un conjunto de mecanismos de comunicación entre nodos que intenta minimizar todo lo posible cualquier problema que pueda surgir en el árbol.

3.5.1. Mecanismo de conexión

Para conectarse a un canal, el nuevo Nodo envía al Bootstrap server un mensaje con el identificador del canal al cual quiere conectarse (mensaje 1 en Fig. 3.14). El Bootstrap tiene que seleccionar un Big-Parent del árbol de ese canal y comunicárselo al nuevo Nodo (mensaje 2). Cuando el nuevo Nodo recibe la respuesta del Bootstrap, envía un mensaje de *join request* (mensaje 3) al Big-Parent asignado, para indicarle que quiere conectarse. El Big-Parent solicitado puede hacer tres cosas: asignarle un nodo de su mismo nivel (si es que hay recursos) decirle al nodo que se conecte a él mismo si tiene recursos o reenviar la petición a otro Big-Parent de otro nivel.

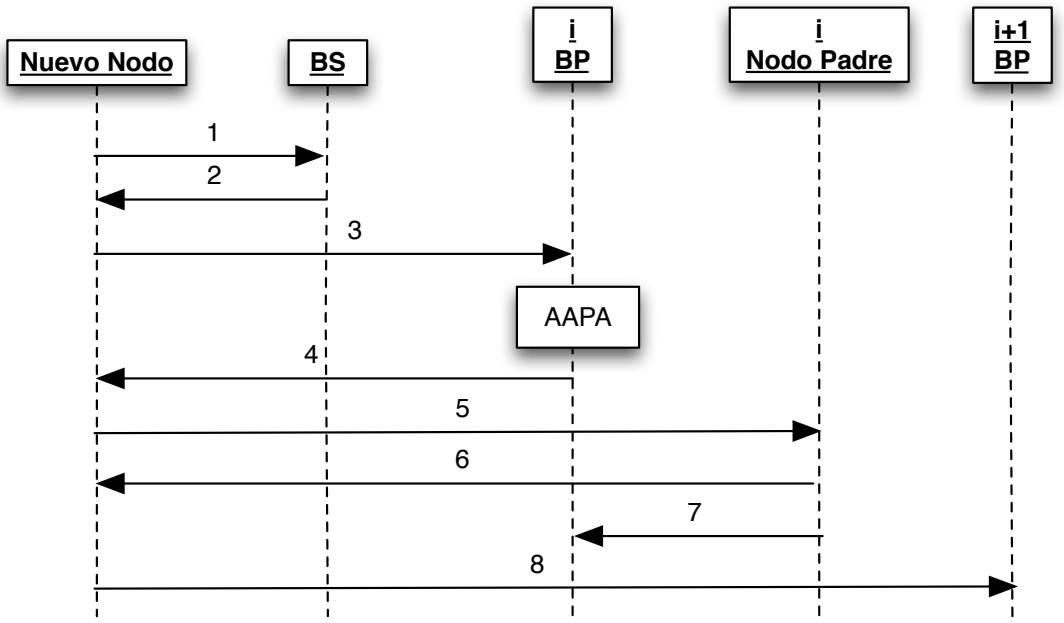

Figura 3.14: Mecanismo de Conexión a un Nodo normal

Si el Big-Parent ha decidido que se conecte a él mismo, le añade a su lista de hijos y

de nodos del nivel inferior. Si no es así, la selección del padre se hace escogiendo el nodo de su propio nivel que menos recursos ocupados tenga, es decir, el que menos nodos hijos tenga. En el caso en el que el Big-Parent detecte que no hay recursos disponibles en su nivel, devolverá al nodo el identificador de otro Big-Parent[2]. En el caso de tener recursos en su nivel, en el mensaje *join reply* que le devuelve al Nodo, el Big-Parent le especifica cuál va a ser su siguiente padre, su nivel i y también el Big-Parent del nivel i.

Cuando el Nodo recibe la respuesta del Big-Parent (mensaje 4), si su padre va a ser el propio Big-Parent (nivel $i-1$), el nodo se considera conectado y manda un mensaje al Big-Parent de su nivel i para informarle de su existencia y de sus recursos. Con esta información, el Big-Parent de nivel i añadirá al nuevo Nodo a su tabla de nodos en el nivel i.

En el caso de que la conexión se haga con un nodo normal, el Nodo *padre* que recibe este mensaje (5) de *join* acepta la conexión del nuevo Nodo *hijo* y lo añade a su lista de hijos. Además, el nodo *padre* envía un mensaje (7) al Big-Parent de su nivel para decirle que tiene un hijo más (un recurso menos), por lo que el Big-Parent lo añade en su tabla de nodos del nivel inferior ($i+1$). Finalmente, el nuevo nodo envía un mensaje (8) al Big-Parent de su nivel para que lo añada a su tabla.

3.5.2. Mecanismo de eliminación de Nodos de un Árbol

Un Nodo se elimina del árbol cuando el usuario abandona el canal o el sistema (mensajes 1 a 5 de la Fig. 3.15). Antes de abandonar el árbol, el Nodo avisa a su padre para que no le siga enviando vídeo (mensaje 1) y a sus hijos para que se reconecten a otro padre (mensaje 2) y, si el nodo que abandona es Big-Parent, avisa al Bootstrap para que marque el nivel y no asignarle temporalmente nodos. En todo este proceso, tanto el padre como los hijos mandan mensajes y utilizan mecanismos para hacer saber a sus Big-Parents el abandono del nodo. Estos mecanismos se verán a continuación.

El nodo padre del que abandona el canal envía un mensaje (mensaje 3 de la Fig. 3.15) al Big-Parent de su nivel para comunicarle que tiene un recurso más. A su vez, este Big-

[2]Una solución similar sería que el Bootstrap server le devuelva al nodo una lista de Big-Parents, por lo que el propio nodo podría elegir otro Big-Parent en caso de que el anterior no devuelva un nodo al que conectarse.

Parent avisa (mensaje 4) al Big-Parent del nivel i que un nodo de su nivel ha abandonado el canal[3] por lo que éste ejecuta el algoritmo de asignación de padres adoptivos (AAPA), ya que pudo haber asignado como padre adoptivo al nodo que ha abandonado el canal. Si fuese así, asignaría nuevos padres adoptivos a los nodos correspondientes (mensaje 5).

En resumen, el padre deberá avisar al Big-Parent de su nivel que tiene un recurso disponible más, y los hijos deben reconectarse lo antes posibles a su siguiente padre, que ha sido asignado previamente.

3.5.3. Mecanismo de reconexión

Al recibir un mensaje *leave* de su Nodo padre, los *hijos huérfanos* de dicho nodo deberán reconectarse enviando un mensaje (mensaje 6 de la Fig. 3.15) *reconnect* al nodo que cada uno de ellos tiene asignado como *padre adoptivo*. En el mensaje de *reconnect* un nodo pide reconectarse indicando además a qué nodo estaban conectados, que será el nodo que ha abandonado el canal.

El nodo que recibe el *reconnect* añade a su nuevo hijo a la tabla e informa (mensaje 8) al Big-Parent de su nivel que un nodo de su nivel se ha ido (el que le ha dicho su nuevo hijo) y también los datos de su nuevo hijo. Con esta información, el Big-Parent de nivel i ejecutará el algoritmo de asignación de padres adoptivos y enviará dicha información a los nodos correspondientes (mensaje 9).

3.5.4. Mecanismo de elección de Big-Parent

El Big-Parent de un nivel i lo elige el Big-Parent del nivel superior $i-1$. El Big-Parent tiene una tabla en la que, aparte de aparecer el identificador de cada nodo, el tiempo que llevan conectados y el padre actual y padre adoptivo que tienen asignados, también aparece el Big-Parent que conoce cada nodo y por supuesto si un nodo es Big-Parent. Para elegir un nuevo Big-Parent en el nivel $i+1$, el Big-Parent de nivel i busca de entre los nodos de nivel inferior aquél que lleve más tiempo conectado, ya que ese Nodo es el que tiene más probabilidad de seguir en el canal ([SGG+02]), disminuyendo, en media,

[3] Con esto se puede reaccionar ante salidas desordenadas de los nodos.

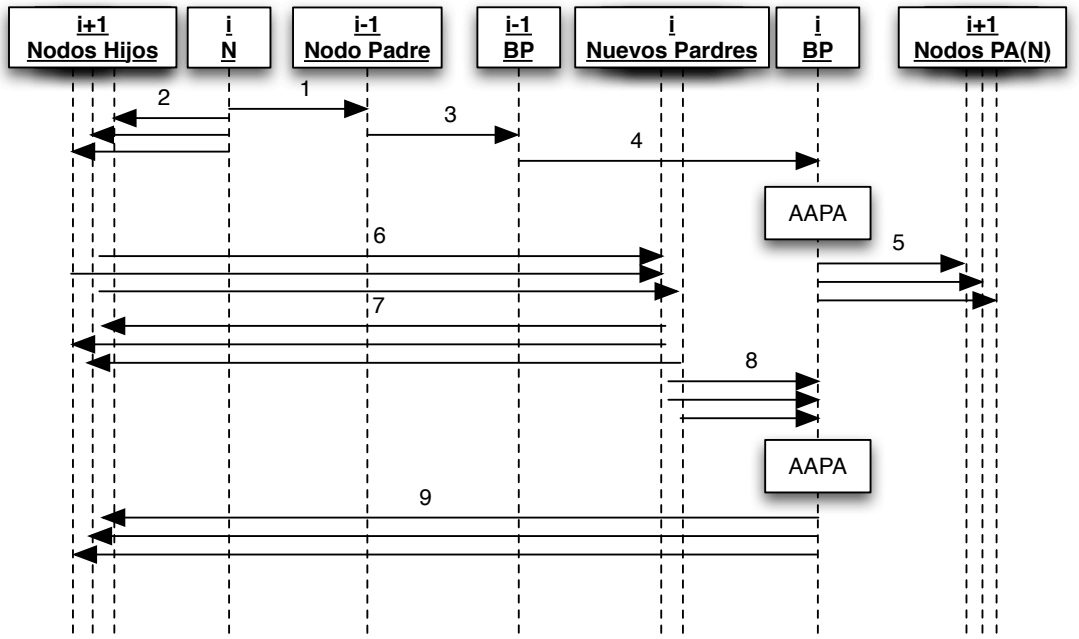

Figura 3.15: Abandono de Nodos normales y reconexión

el número de mensajes necesarios para la elección y posterior notificación de los Big-Parents.

Con esa información, el Big-Parent de nivel i envía un mensaje a todos los nodos del nivel inferior indicándoles cuál ha sido el Nodo elegido para ser Big-Parent del nivel $i+1$. Cuando cada nodo del nivel inferior reciba el mensaje deberán enviar un mensaje al nuevo Big-Parent con sus tablas de hijos, para que éste conozca toda la información de su nivel y del nivel inferior, incluido el Big-Parent del siguiente nivel. El nuevo Big-Parent también debe enviar un mensaje al Bootstrap para sustituir al que ya había en su nivel e indicarle que ya puede aceptar peticiones de conexión.

3.5.5. Mecanismo de asignación de Padre Adoptivo

Los padres adoptivos de cada nodo de un nivel i los elige el Big-Parent del nivel $i-1$. La elección se hace de manera aleatoria de entre todos los nodos que tenga el Big-Parent en su nivel, incluído él mismo. Esto se hace así porque a priori no es fácil determinar cuál es el mejor nodo que servirá de padre adoptivo. El padre adoptivo se asigna en el

momento de la conexión, en las reconexiones y cuando se produce un abandono de un nodo y hace falta actualizar esa información en los nodos que tuvieran asignado como padre adoptivo al nodo que se acaba de ir.

3.5.6. Mecanismo de asignación de Big-Parent en el Bootstrap

Cuando un Nodo se conecta al Bootstrap para obtener un punto de conexión al árbol de un canal, el Bootstrap busca un Big-Parent de forma pseudo-aleatoria de entre los niveles que estén disponibles (que serán todos aquellos cuyo Big-Parent no se haya ido del canal y el nivel se esté reconfigurando todavía), y se lo asignará al nuevo Nodo. Se dice que el algoritmo de búsqueda de Big-Parent es pseudo-aleatorio porque se tiene en cuenta cuántos nodos se han conectado desde el último Big-Parent asignado, no utilizando este último hasta que el número de nodos nuevos haya sobrepasado un cierto límite. Este mecanismo permite controlar la altura de los árboles, ya que se elimina la posibilidad de que nodos consecutivos que entren al canal se aniden formando una rama, sin más nodos en sus niveles correspondientes.

3.5.7. Mecanismo de abandono de canal y reestructuración del árbol

Cuando un Nodo abandona el canal, lo único que tiene que hacer es enviar un mensaje *leave* a su padre y a cada uno de sus hijos. Además, si el nodo es Big-Parent, envía un mensaje al Bootstrap, para que éste sepa que provisionalmente no hay Big-Parent en el nivel y no asigne nodos ahí hasta que se haya reestructurado. A partir de ese momento, el nodo es libre de abandonar ese canal.

El problema puede surgir, evidentemente, en los nodos que se quedan. Los hijos, al recibir el mensaje *leave*, se reconectan a su padre adoptivo siguiendo el mecanismo de reconexión descrito anteriormente.

Cuando el padre recibe el *leave*, envía un mensaje al Big-Parent de su nivel indicándole el nodo que se ha ido. El Big-Parent, de esta manera sabe que el nodo tiene un hijo menos, y además si el que se ha ido fuera Big-Parent, tendría que buscar un nuevo Big-Parent y enviar mensajes según el mecanismo de elección de Big-Parent visto anteriormente. Si el que se ha ido no es Big-Parent, envía un mensaje al Big-Parent del nivel inferior para que éste actualice la tabla de nodos en su nivel.

3.5.8. Mecanismo de resolución de eventos concurrentes

El problema que plantea diseñar un protocolo como este es, para el caso que nos interesa, saber manejar correctamente los estados del sistema dependiendo de los mensajes que reciba cada entidad. En este caso, los nodos pueden recibir muchos tipos de mensajes dependiendo del rol que representen o que puedan llegar a desarrollar, y tienen que estar preparados para almacenar la información que se les envíe de manera correcta y coherente. Dicho de otra manera, cada nodo tiene que saber en qué estado está y cómo tiene que reaccionar ante todas las situaciones posibles, dependiendo de los mensajes que reciba. Es muy importante mantener las tablas lo más actualizadas posibles y detectar desconexiones tanto de nodos hijos como del nodo padre, para actuar consecuentemente y no perder información que pueda dar lugar a saltos en la reproducción del vídeo, o a una mala administración del nivel, en el caso de los Big-Parent.

En cuanto al mecanismo de resolución de eventos concurrentes que se plantea en esta sección, se refiere a la manera de actuar de un nodo cuando está en la situación de recibir diferentes tipos de mensajes con la misma o diferente información, pero que le hacen actuar de una manera concreta. Este mecanismo se utiliza principalmente para la generación de Big-Parents y para las desconexiones.

- Vimos en la sección de mensajes que un Big-Parent puede enterarse de que tiene un nodo menos en su nivel tanto a través del padre del nodo que se va como a través de los hijos que tuviera. Pero es difícil predecir de qué forma se enterará antes, ya que depende del estado de la red y de cómo estén interconectados entre sí los nodos que le hagan llegar esa información. Por tanto, un Big-Parent debe estar preparado para recibirla y procesarla de manera correcta, obviando redundancias (necesarias, por otra parte, por fiabilidad), pero manteniendo coherente la información de los nodos de su nivel y la de los nodos del nivel inferior. De esta manera, cuando un Big-Parent detecta desconexiones de nodos en su nivel, actúa de manera ordenada para recomponer la información y, por ejemplo, reasignar un nuevo padre a los nodos que tuvieran como padre adoptivo al nodo que se ha ido, y al mismo tiempo, sólo realizar esta acción una vez, independientemente de las fuentes por donde reciba la información.

  En la Fig. 3.16 se puede ver el problema de la concurrencia en el abandono de un

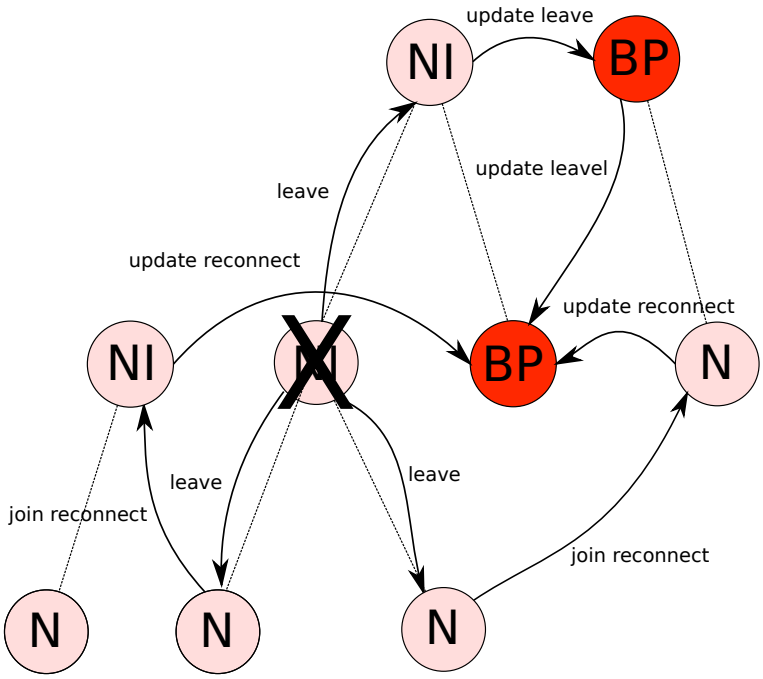

Figura 3.16: Concurrencia de mensajes en desconexión

nodo y los mensajes que produce.

- En el caso de la asignación de un nuevo Big-Parent, un nodo se convierte en Big-Parent cuando recibe un mensaje *update new bp* en el que se le dice que va a serlo. También es posible que un nodo del mismo nivel reciba antes la información de quién es el nuevo Big-Parent, y le envíe un *update resources* antes de que el nuevo Big-Parent reciba la notificación. Aún así, el nodo debe ir reteniendo y organizando esta información para empezar a asumir su nuevo rol y finalmente convertirse en Big-Parent.

  En la Fig. 3.17 se observan los mensajes que podría producir el abandono de un Big-Parent, y la situación (llegada de mensajes) a la que se enfrentaría el nodo que va a ser el nuevo Big-Parent del nivel (en tonalidad más oscura que el resto)

Por estos motivos se ha prestado especial atención al tratamiento de los estados de un Big-Parent, para mantener la coherencia en el árbol y hacer que la arquitectura sea estable, siendo capaz de reestructurarse lo más rápido posible frente a la salida de nodos de la red, tanto si se producen de manera ordenada como desordenada. Todas estas eventua-

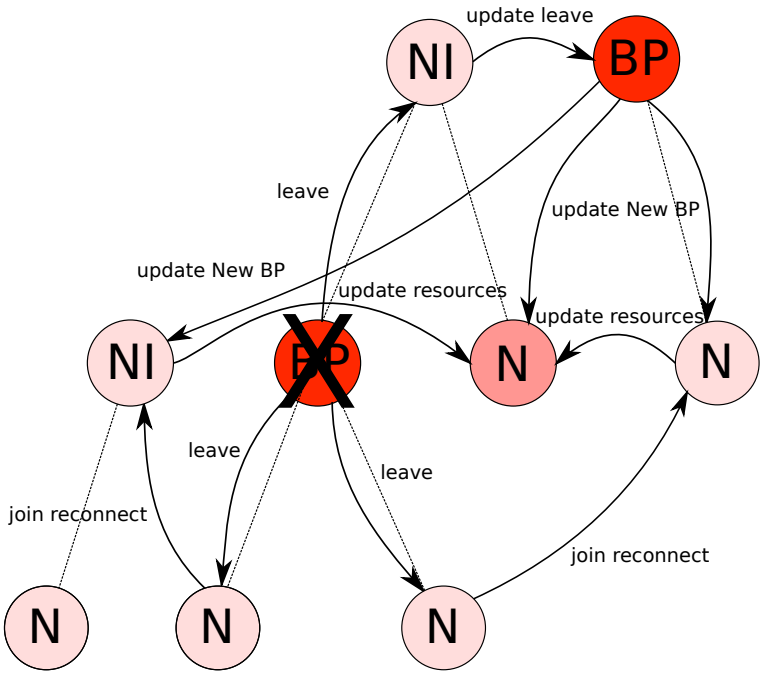

Figura 3.17: Concurrencia de mensajes en la asignación de un nuevo Big-Parent

lidades que pueden parecer lógicas y triviales en su explicación, en realidad son bastante complejas de administrar y detectar, siendo altamente dependientes de la naturaleza del protocolo, y por ello se ha puesto especial cuidado a la hora de diseñarlo.

3.5.9. Mecanismo de temporizaciones

Para el diseño de este protocolo se ha tenido en cuenta la posibilidad de que existan eventos retardados, puesto que por la propia naturaleza de la red se producen esperas debido a los tiempos de tránsito de los paquetes y a los diferentes retardos de transmisión que puedan existir. Por tanto, parece lógico esperar en determinadas situaciones hasta que se produzca un evento para realizar alguna acción. Por supuesto, estas esperas no son activas, por lo que se puede seguir el flujo normal del programa.

Se han incorporado temporizaciones o mecanismos de espera en las conexiones, en los abandonos/reconexiones y para la generación de Big-Parents.

- En el caso de las conexiones, los nodos esperan un tiempo para ponerse en contacto con los Big-Parents o padres y poder conectarse. Si pasado ese tiempo no lo han conseguido, vuelven a iniciar los mecanismos de conexión donde corresponda.

En el caso de los abandonos, los nodos envían el mensaje de reconexión (según el mecanismo de reconexión visto anteriormente) y esperan un tiempo a que se produzca. Si no lo consiguen vuelven a repetir el mecanismo de conexión.

- Cuando un nodo es nombrado Big-Parent, debe esperar un tiempo para recibir la información de los nodos de su nivel y recomponer las tablas con los recursos disponibles. Ese tiempo depende de cuánto tarde en recibir todos los mensajes, y como a priori no sabe cuántos nodos hay en su nivel, debe esperar un tiempo razonable para considerarse estable y empezar a actuar correctamente como Big-Parent. Esto no quiere decir que no pueda estar realizando las tareas propias de un Big-Parent, sino que durante ese tiempo no informará al Bootstrap de que se ha convertido en Big-Parent y no pondrá en práctica el mecanismo de elección de Big-Parent (del siguiente nivel), esperando encontrarlo en los mensajes *update resources* de los nodos de su nivel.

3.6. Resumen

En este capítulo se han detallado las características más importantes del sistema, los mensajes y el protocolo. Se ha realizado un diseño completo de la arquitectura, cuidando todos los detalles posibles para poder llevar a cabo una implementación completa de la misma. Además, se ha puesto especial cuidado en la descripción de los aspectos más importantes del protocolo y de los mecanismos que permiten a los nodos comunicarse entre sí. Y también se ha intentado poner énfasis en las características que mejorarán el sistema y que servirán para cumplir los objetivos propuestos. En el siguiente capítulo se verán los resultados obtenidos de haber realizado una implementación de este diseño.

Capítulo 4
Validación

4.1. Simulador

Para comprobar el correcto funcionamiento de la arquitectura diseñada, se ha diseñado a su vez un simulador basado en eventos que permita simular una red con los mensajes pertinentes. Ambas cosas se han implementado con Java [1].

4.1.1. Características

A continuación se detallarán las características y ventajas de este tipo de simulador.

- Orientado a Eventos: al ser un modelo orientado a eventos, el tiempo en el que se generan mensajes y se procesan no es "tiempo real", sino que el tiempo avanza hasta que se produce un cambio de estado en alguno de los elementos que componen la red. A este cambio se le denomina evento. La ventaja de este tipo de simulaciones es que permiten reducir notablemente el tiempo en el que se obtienen resultados. Se puede simular el transcurso del envío de mensajes de un día entero en unos pocos minutos.

- No hace falta distribuir los nodos en una red real: otra ventaja importante de realizar una simulación es que la red también se simula. Todos los nodos pueden estar ejecutándose en el mismo ordenador, y así es más fácil controlar los estados y eventos de cada uno de los nodos que componen la red. Además, resulta muchísimo más económico ya que se necesitan menos recursos para estudiar su comportamiento.

- La información se recopila más fácilmente: al estar todos los nodos en el mismo ordenador, los resultados que produzca cada uno de ellos quedarán almacenados

[1] http://java.sun.com

en el mismo sitio, permitiendo ver rápidamente la situación de la simulación en un momento dado. Podemos decir que es un entorno más controlado, en el que se simplifica bastante la detección de errores.

Parece que realizar una simulación tiene ventajas notables frente a hacer una emulación en un entorno de red real. Pero también presenta algunas dificultades, que son las que ha habido que solucionar a la hora de diseñarlo correctamente.

- Por un lado está el problema de la conciencia del tiempo: cada nodo que se encuentra en la red virtual debe conocer el tiempo en el que entra, el tiempo que tarda en salir o el tiempo que tiene que pasar hasta que hay que realizar alguna acción. A su vez, debe enviar mensajes a otros nodos que tardarán un tiempo en enviarse y recibir de cualquier nodo sin importar el orden de recepción, como en una red real. Debe haber algún elemento que sea capaz de proporcionar esa conciencia del tiempo a todos los nodos. y que para todos sea la misma en el momento en el que se realiza alguna acción.

- Por otro lado existe el problema de la concurrencia de eventos en el tiempo: hemos dicho que cada nodo puede generar y recibir mensajes en cualquier instante de tiempo, para parecerse lo más posible a un entorno real. Eso quiere decir que puede haber coincidencia en los instantes de tiempo en el que se vaya a producir alguna acción. Y esa situación debe estar contemplada y controlada en la medida de lo posible.

- Comunicación entre nodos: En una red real, los nodos se envían mensajes entre sí a través de una pila de comunicaciones que es transparente para ellos, al realizar esa función el sistema operativo. Pero aún así, los nodos tienen que construir los mensajes y enviarlos. Y el sistema operativo debe ser capaz de localizar los destinatarios e intentar que los mensajes les lleguen. El simulador debe proporcionar un mecanismo parecido que abstraiga de esos problemas a los nodos.

- Monitorización y recopilación: Hemos visto que el hecho de simular una red tiene como ventaja tener más facilidad de recopilar información debido a la concentración de todos los participantes en el mismo ordenador. Eso es totalmente cierto, pero hay que saber aprovechar esa posibilidad para sacar partido a la información que se recopila. Para ello hay que diseñar un mecanismo de control de la simulación que no sólo nos permita obtener fácilmente los datos de salida, sino que además

permita detectar fácilmente los fallos que se produzcan a través de un sistema de trazas o algo parecido.

Con todas estas características y condiciones vamos a ver cómo se ha realizado la implementación.

4.1.2. Implementacion

En primer lugar, surge la necesidad de tener un eje temporal. Una estructura donde se puedan ir añadiendo los eventos según vayan surgiendo y que se respete el orden temporal de los mismos. Además, es necesario que si varios eventos ocurren en el mismo instante de tiempo, éstos no se solapen y se procesen todos. Lógicamente el procesado de los mismos se hará de manera secuencial. Pero digamos que podría equivaler a una cola de eventos. Además, estos no tiene por qué ir destinados al mismo nodo.

También hay que plantearse el problema de la granularidad del tiempo. Mientras que los tiempos de estancia en el canal van a tener una granularidad de segundos, los tiempos de envío de mensajes o de esperas de los nodos tienen que ser del orden de milisegundos, si se quiere simular una red realista. Por lo tanto, se ha elegido un eje de tiempos de precisión de milisegundos. No hay problema en la duración de la simulación puesto que un dato de tipo *long* puede abarcar varios años contados en milisegundos, probablemente más que el número de eventos que pudiera almacenar la memoria del sistema.

La estructura de datos utilizada es una tabla hash en la que en cada entrada se almacenan vectores de eventos. Y cada entrada es un objeto Long que representa el tiempo en el que se produce el evento. Es decir, que si en el milisegundo 3576 hay un evento programado, se extraerá de la tabla hash el vector de eventos ubicado en la posición registrada por la key 3576, y los eventos que haya en dicho vector se procesarán de manera secuencial por orden de adhesión. El vector de eventos se ha estructurado en lo que se denomina un Piscina de Eventos (PoolEvents).

La tabla hash se recorrerá de manera secuencial desde el principio de la simulación hasta el momento decidido como final. Es decir, que si se ha decidido que la duración de la simulación va a ser de un millón de milisegundos, habrá un contador que se recorra todos

esos milisegundos y en cada uno hará una búsqueda en la tabla hash por si en la key correspondiente a ese milisegundo existe un Evento o conjunto de eventos para procesarlos.

Pero, ¿qué es un evento?. Un evento está formado por un nodo origen, un nodo destino y un tipo, el cuál representa el tipo de mensaje que se procesa en el evento. La manera de enviar un evento es enviar un clon del nodo origen a la referencia del nodo destino, indicando el tipo del evento o mensaje. Cuando se procesa, el nodo destino realiza las acciones necesarias dependiendo del tipo de mensaje y utilizando los datos suministrados por la copia del nodo origen. Es importante que sea una copia y no el nodo original, porque su estado puede variar dinámicamente, dependiendo del momento en el que se produce el evento, y la información recopilada por el nodo destino podría no ser la correcta. Asimismo, es importante que el nodo destino sea la referencia del objeto original, puesto que es cada nodo propiamente el que tiene que procesar la información. Además, esa es la manera que tiene cada nodo de conocer a los demás nodos. Cada nodo tiene una variable que se representa a sí misma, y que perdura en las clonaciones. Tener esa referencia es el equivalente al método de localización en una pila de comunicaciones estándar.

La manera de añadir eventos al simulador es mediante el manejador de eventos (EventHandler). Este elemento se encarga de introducir en la tabla hash el PoolEvents que produce un nodo. Además, a través de él se asignan los tiempos de envío entre mensajes. Un nodo puede producir un conjunto de mensajes (eventos) destinados a diferentes nodos destino. Estos se guardan en un PoolEvents origen del que se van extrayendo para distribuirlos por los PoolEvents que correspondan a los tiempos de envío de cada mensaje. En la Fig. 4.1 se puede ver más claramente cómo funciona.

Los nodos generan un PoolEvents que contiene un conjunto de eventos que son analizados independientemente para introducirlos en el eje de tiempo según corresponda. Esto dependerá del origen y destino del evento, y representará el tiempo de transmisión del paquete entre ambos. A la hora de procesarse, se coge el PoolEvents de cada instante de tiempo y ejecuta cada evento.

Para realizar la monitorización, cada Nodo tiene acceso a un objeto Monitor, que es el que se encarga de registrar los eventos, y en cualquier momento puede acceder a él para guardar la información que quiera. Se distinguen dos tipos de eventos a monitorizar.

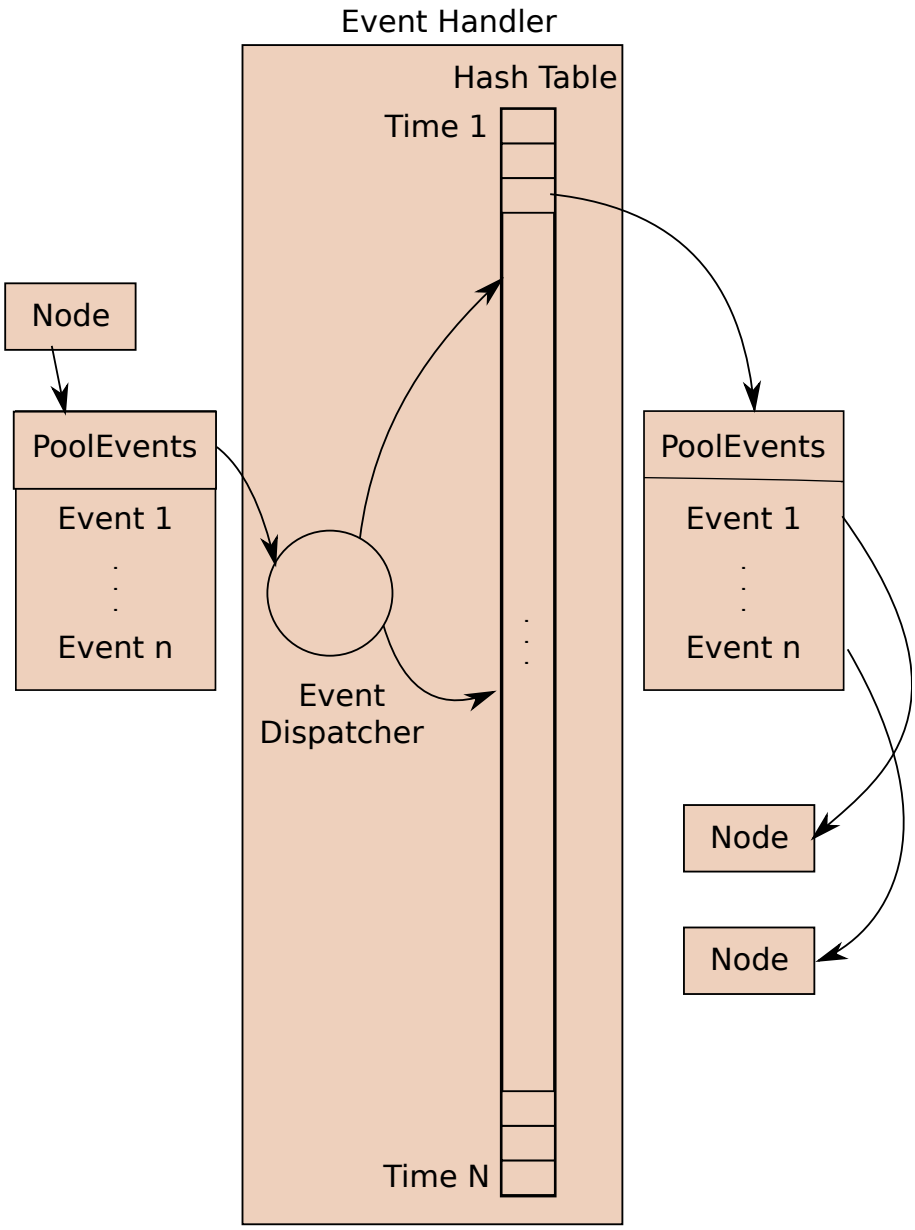

Figura 4.1: Funcionamiento del manejo de eventos

- **Eventos de Nodo:** son los que se producen dentro de un nodo. Es decir, un nodo es capaz de monitorizar su estado en cualquier momento, escribiendo sus variables a través del objeto Monitor. Un ejemplo de evento de nodo a monitorizar sería cuando se produce una desconexión. En ese caso, el nodo puede dejar constancia de su estado (tiempo que llevaba conectado, canal en el que estaba, hijos que tenía...) y es interesante recopilar esa información para hacer análisis posteriores.

- **Eventos de Árbol:** son eventos en los que se recopila información de todos los nodos del árbol a la vez, para conocer por ejemplo la forma que tiene, número de nodos por nivel, o conocer cualquier parámetro global común a todos los nodos, tal como el estado o el tipo. La manera de conseguir la información en estos eventos es mediante búsqueda recursiva dentro del árbol. Hay que recordar que al ser una simulación, el árbol que se generaría físicamente como la conexión de cada una de los usuarios, en este caso es una estructura de datos en forma de árbol normal y corriente.

Por último, en la Fig. 4.2 se muestra un diagrama simplificado de las clases utilizadas para desarrollar el proyecto. Se puede ver que casi todo gira en torno a la clase Nodo, que constituye el núcleo del sistema, ya que es el que lleva la mayor parte de la lógica implementada, tanto a nivel de mecanismos, como de protocolos y estados.

4.2. Resultados Obtenidos

Como se dijo en la sección anterior, para realizar la validación de la arquitectura diseñada se ha implementado un simulador basado en eventos que es capaz de simular una red con los mensajes que se envían entre los nodos y permite estudiar el comportamiento de éstos frente a cambios que se produzcan en el árbol, en especial las conexiones y reconexiones que se produzcan, y que son las que van a centrar las mediciones realizadas.

Gracias a la implementación de este simulador se podrán obtener datos de árboles de un tamaño considerable y que se podrán procesar más fácilmente. Por ello, al simulador se le ha dotado de un sistema de monitorización que permite obtener un gran número de datos del estado del árbol o de los nodos.

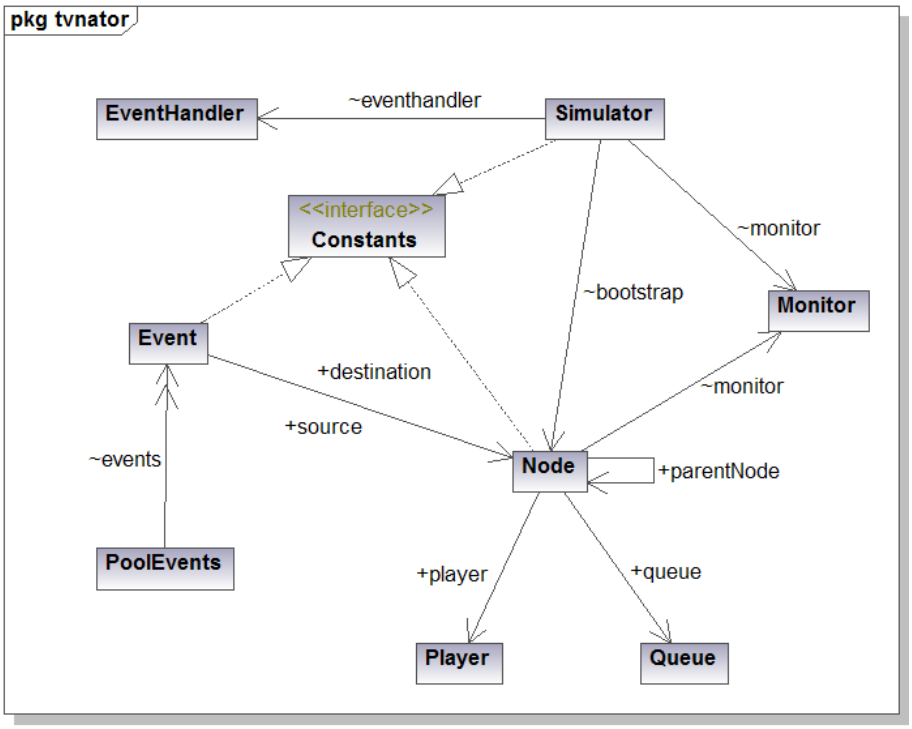

Figura 4.2: Estructura de clases

Para que este simulador sea válido, debe reproducir lo más fielmente posible las condiciones de una red real y el comportamiento de los usuarios para los que se ha diseñado la arquitectura.

- Para conseguir las condiciones de una red real, los mensajes que envíen los nodos deben estar sujetos a retardos parecidos a los que se encontrarían en Internet. Es decir, que para cada par de nodos debería existir un retardo diferente que estará en función de la localización de cada nodo en la red. Nuestra solución está pensada para ser ejecutada en Internet, pero para intentar reproducir las condiciones completas se necesitaría una simulación muy compleja. Por lo tanto se ha optado por hacer una aproximación basada en datos empíricos obtenidos a través del proyecto PingER[2], el cual realiza estadísticas de retardos entre diferentes puntos de Internet, y que permite elegir varios parámetros para sacar las medidas. En concreto, nosotros nos hemos basado en la tabla resumen del RTT medio para nodos localizados en Europa, y para un tamaño de paquete de 100 bytes. Hemos decidido utilizar datos de un sólo continente porque la probabilidad de que los usuarios que estén

[2]http://www-iepm.slac.stanford.edu/pinger/

viendo una retransmisión en directo sean de una zona concreta es bastante alta. En base a estos datos se ha generado una variable aleatoria normal con media $19{,}8ms$ y desviación típica de $16ms$ para cada enlace entre cada par de nodos. La elección del tipo de variable aleatoria se ha hecho por aproximación a partir de histograma realizado con los datos extraídos del mes de Marzo de 2010 del citado proyecto.

- Otro dato de entrada importante es el tiempo de estancia de cada usuario en un canal cualquiera. Este dato lo hemos extraído de [CRC+08] en donde se obtuvieron diferentes parámetros de un servicio desplegado de IPTV. Aunque el sistema analizado en dicho artículo está basado en una infraestructura dedicada para IPTV desplegada utilizando multicast IP, nosotros tomamos el valor de tiempo de estancia en el canal como referencia, ya que nuestro objetivo es brindar la misma experiencia a los usuarios del sistema P2P propuesto que la que perciben los usuarios de un sistema dedicado. Ya que la información proporcionada en [CRC+08] es agregada entre todos los canales ofertados, hemos decidido simplificar el análisis y aplicar la misma distribución de tiempo de estancia a todos los canales simulados. Por este mismo motivo, todos los canales y usuarios tendrán el mismo comportamiento, por lo que los usuarios tendrán un tiempo de estancia en el canal siguiendo la misma distribución y, una vez abandonen su canal actual, elegirán con la misma probabilidad (siguiendo una distribución uniforme) otro canal.

Con todos estos datos que intentan aproximar una red con condiciones reales, se ha creado una simulación en la que existe un Bootstrap server en el que están registrados 4 canales, y que cuenta con un número total de 400 usuarios en el sistema. Aunque existen otras condiciones y variables importantes como son el número de recursos (ancho de banda disponible en uplink) y tamaño de buffers en cada equipo, en estas simulaciones se supone que esto no es problema, asumiendo un número infinito de recursos.

4.2.1. Estructura de los árboles

Una parte importante de la validación es comprobar que los árboles de distribución se crean y reconstruyen de manera adecuada. Para ello hemos obtenido los parámetros típicos de un árbol de distribución para comprobar la morfología media resultante. En la Fig. 4.3 se representan los valores medios y desviación típica del porcentaje de nodos por cada uno de los niveles. Un resultado interesante es que, aún teniendo número ilimitado de recursos en cada nodo, los nodos padre no tienen muchos nodos hijos conectados ya

que la relación entre nodos de cada nivel no es muy grande. Además, aunque el número de nodos por nivel no es constante, sí se puede observar que se distribuye entre los diferentes niveles. Lógicamente, por el orden de conexión, el porcentaje de nodos de los niveles inferiores comienza a decrecer a partir de un cierto punto.

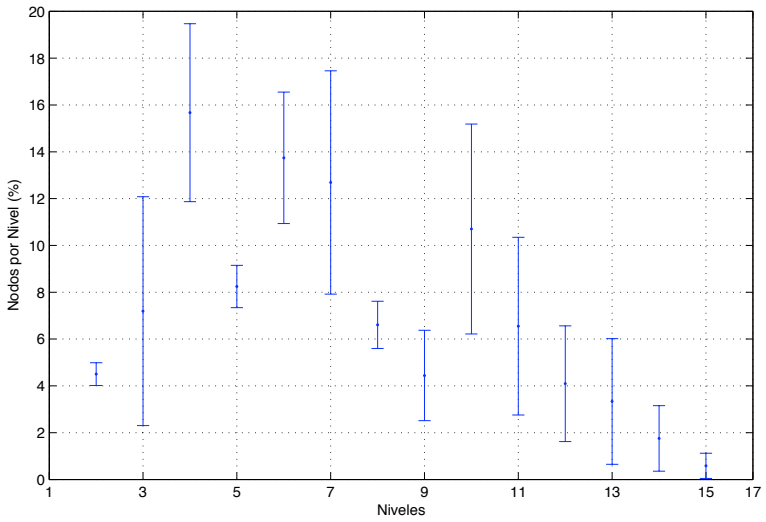

Figura 4.3: Porcentaje de nodos por nivel

Otro parámetro interesante es el del número de nodos hoja en el árbol. Tras realizar varias simulaciones, hemos obtenido que el número medio de nodos hoja es de $33{,}5\,\%$ del total de nodos y una desviación típica de $1{,}75$. Además, hemos estimado que el número medio de niveles es de $13{,}1$ con una desviación típica de $1{,}25$. Estos valores nos demuestran que el árbol se encuentra acotado tanto en altura como en el número de nodos hijo de cada nodo.

4.2.2. Tiempos de conexión y reconexión

Uno de los valores más importantes, y motivo por el cual hemos realizado esta propuesta, es el tiempo que necesita un nodo para reconectarse una vez detecta que su padre actual se ha desconectado. Para estimar dicho valor, se han realizado 30 ejecuciones diferentes de 1 día de duración cada una de ellas, obteniendo los resultados que se muestran en la Fig. 4.4. En esa figura se puede ver la función de distribución (Cumulative Distribution Function o CDF en inglés) para los tiempos de conexión (tiempo necesario para que un nodo se conecte o cambie a otro canal) y de reconexión (tiempo necesario para

que un nodo hijo se reconecte a otro nodo padre). De estos resultados, se desprenden las siguientes conclusiones:

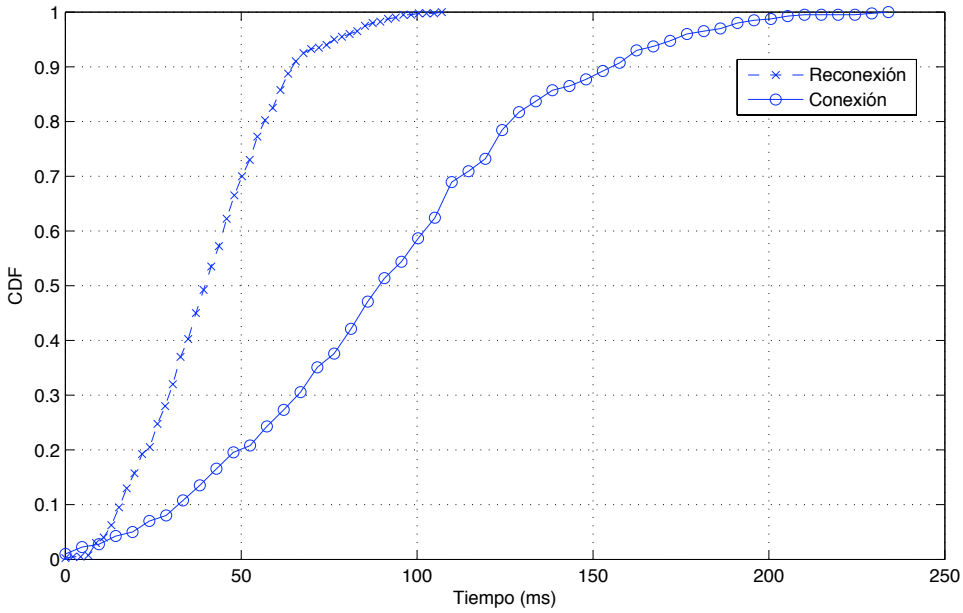

Figura 4.4: CDF de los tiempos de conexión y reconexión

- Para el tiempo de conexión, el valor máximo obtenido es de $234ms$ y el mínimo de $35ms$. El valor medio obtenido entre todas las realizaciones es de $115ms$ con una desviación típica de $38{,}5ms$.

- Para el tiempo de reconexión, el valor máximo obtenido es de $107ms$ y el mínimo de $10ms$. El valor medio de todas las reconexiones es $48ms$ con una desviación típica de $17{,}3ms$

Lo más importante es que el tiempo medio de reconexión de un nodo es muy pequeño, e incluso el valor máximo es aceptable para que un nodo no pierda muchos paquetes durante esta fase de reconexión. Lógicamente, el tiempo de reconexión será más pequeño que el de conexión por la utilización de padres adoptivos.

Analizando detalladamente los resultados, se ha podido comprobar que tan sólo el $0{,}45\,\%$ de las reconexiones de los nodos no se realizaba correctamente con sus *padres adoptivos*, sino que debían contactar con el Bootstrap server.

4.2.3. Carga de los Big-Parent

Otra de los valores más importantes, y que hace muy interesante la propuesta, era distribuir la carga de los nodos que se encargan de administrar el sistema. De esta manera, cada Big-Parent o el Bootstrap server recibirán o enviarán una cantidad de mensajes significativamente menor que el que recibiría o enviaría un nodo dedicado en un sistema centralizado. Para comprobar que esto es así, se ha medido el número medio de mensajes que envían y reciben los nodos Big-Parent por minuto, y además, para cada nivel. Los resultados de esas mediciones se muestran en las Fig. 4.5 y Fig. 4.6.

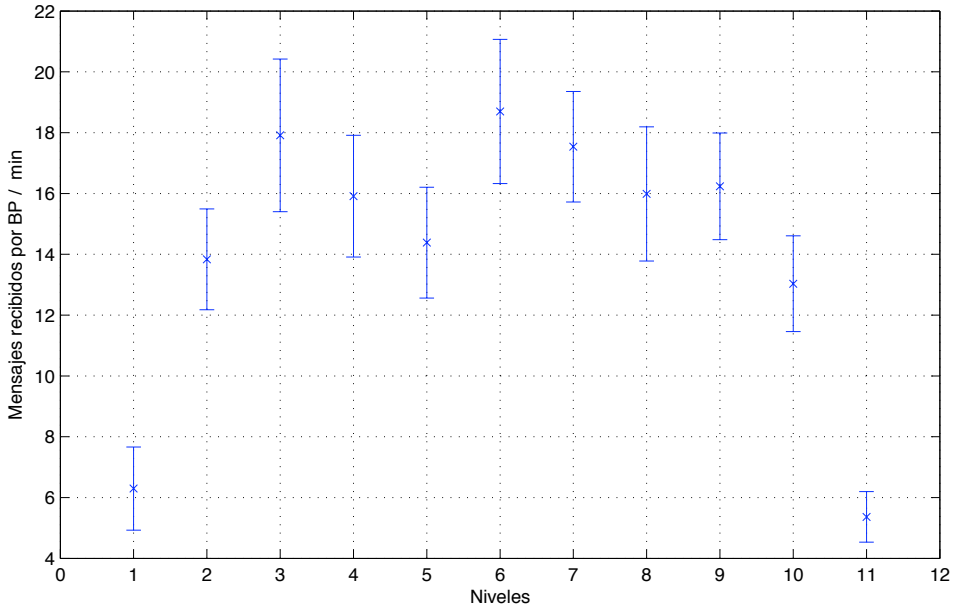

Figura 4.5: Número medio de mensajes de control recibidos en un Big-Parent por minuto y por nivel, con intervalos de confianza del 95 %

Se puede ver que la carga media es bastante uniforme en los niveles medios del árbol, siendo además bastante baja. Este resultado valida nuestro objetivo de distribuir la carga de administración, ya que se ha conseguido que se reparta el número y procesamientos de mensajes de administración del árbol lo mejor posible. También se ve que el número medio de mensajes que envían es aproximadamente el doble de los que reciben. Esto es lógico, puesto que la función que tienen que realizar los Big-Parent es la de administrar en función de los datos que reciban. Si comparamos estos datos con los de la carga del Bootstrap, que resultaron ser de 75 mensajes recibidos y 56 mensajes enviados, en

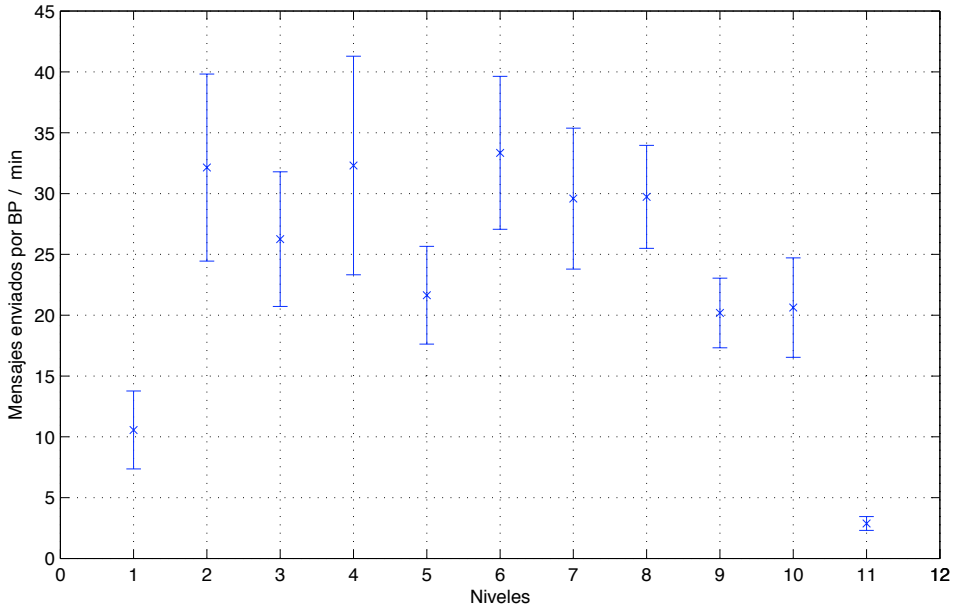

Figura 4.6: Número medio de mensajes de control enviados por un Big-Parent por minuto y por nivel, con intervalos de confianza del 95%

media, vemos que aunque el Bootstrap sigue siendo el que más mensajes procesa en proporción a los Big-Parent de cada nivel, si se sumasen los mensajes de todos los niveles, la carga del Bootstrap sería del 32% del total de mensajes de control para los mensajes recibidos y del 18% para los enviados.

4.2.4. Estabilidad de los Big-Parent

Por último, otro dato muy interesante obtenido es el que se refiere a la estabilidad de los Big-Parent. Es decir, a la probabilidad de que haya que cambiar de Big-Parent en un nivel. Cuantos menos cambios haya que hacer, más estable será el árbol, ya que no se tendrá que reconfigurar tantas veces. Por ese motivo, es muy importante elegir correctamente el Big-Parent de un nivel. Como se describió en 3.5.4, la manera de asignar un Big-Parent en un nivel es elegir aquel nodo que llevase más tiempo conectado. Este tipo de elección ha demostrado ser bastante estable, como se puede observar en la Fig. 4.7, en la que se muestra el número de desconexiones de Big-Parent cada minuto durante los 30 primeros minutos de la simulación.

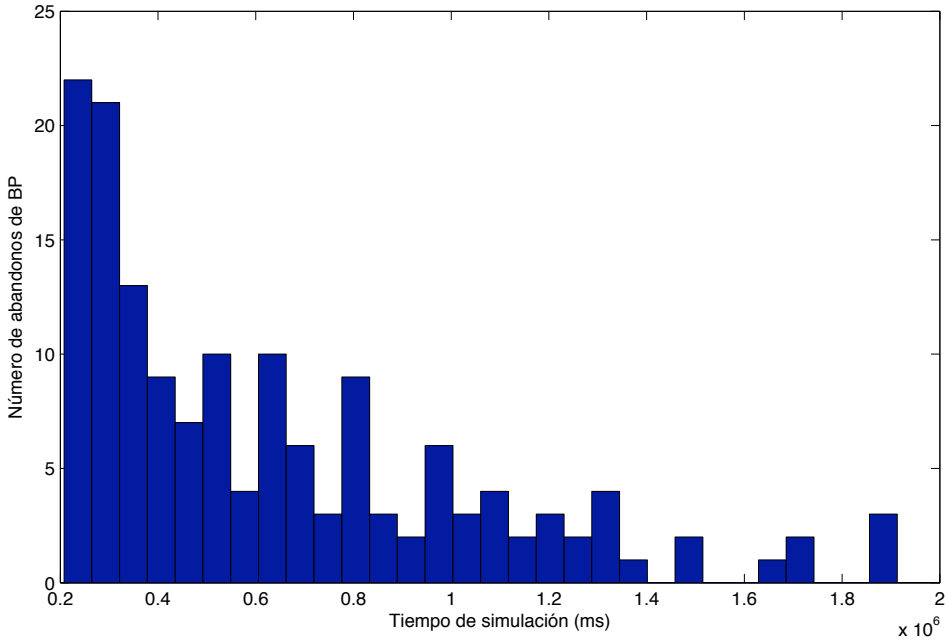

Figura 4.7: Evolución del número de abandonos de Big-Parents

Para entender esta figura hay que explicar el contexto en el que se produce, que es el que interesa hacer notar. Se ha realizado una simulación en la que los valores iniciales corresponden a abandonos producidos por nodos que no han sido elegidos por el sistema, sino que han sido fruto de la creación del árbol (se genera un nivel más en el árbol que necesita un Big-Parent, y el primer nodo que exista al crear ese nivel se convierte en Big-Parent). A medida que pasa el tiempo, según se van desconectando esos Big-Parents, se van sustituyendo por otros elegidos por el sistema y que hacen que poco a poco vaya disminuyendo el número de abandonos que se producen de Big-Parents, hasta estabilizarse en un valor bastante razonable.

4.3. Conclusiones del capítulo

En este capítulo se han realizado una serie de medidas para obtener resultados que validen la propuesta planteada. Para ello, se ha diseñado un entorno de simulación completo en el que poder ver la evolución del ALM y que permita obtener el mayor número de datos posibles. Este entorno es flexible, en cuanto a que se puede adaptar a las condiciones de la red, escalable, ya que se puede ir ampliando la información a recabar con

él fácilmente, y válido, en el sentido de que sirve para analizar los datos que se han considerado necesarios para comprobar el funcionamiento del sistema. Además, para que este entorno simulado sea lo más fiel posible a uno real, se han utilizado aproximaciones empíricas basadas en datos reales de Internet y en el comportamiento de los usuarios.

Con todo esto se ha podido ver que las medidas realizadas cumplen ampliamente con las expectativas, ya que se ha conseguido un sistema que minimiza las pérdidas de datos por reconexión y que sirve para descongestionar el tráfico que se produciría en un sistema centralizado. Además, mediante el análisis de ciertos parámetros y su tratamiento se ha conseguido optimizar el árbol para mejorar su estabilidad en cuanto a nivel de control se refiere.

Por todo esto, consideramos que la propuesta realizada queda validada para los objetivos que se habían planteado.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

A lo largo de este proyecto se ha explicado cómo sería el funcionamiento y mantenimiento de un ALM distribuido que sirva para transmitir vídeo en directo, en concreto para servicios de IPTV. El modelo planteado es sencillo y escalable. Además, está orientado a minimizar el tiempo de reconexión de los nodos y a distribuir la carga de control de los nodos en lugar de que lo haga uno sólo de manera centralizada. En esos dos aspectos fundamentales, la solución propuesta es suficientemente válida, sobre todo a raíz de las pruebas realizadas.

También se ha podido comprobar la estabilidad de los árboles que se crean, haciendo que los Big-Parent cambien con la menor frecuencia posible en cada nivel y que puedan detectar abandonos lo más rápidamente posible para reestructurar el árbol adecuadamente. Esto se consigue gracias a los mecanismos que hacen que se pueda informar tanto por parte del padre del nodo que se va como de los hijos, y también que mejore la rapidez debido al ajuste dinámico de las temporizaciones en función de los RTTs.

Además el protocolo diseñado para la administración es muy escalable, permitiendo añadir más información fácilmente en los mensajes que permitan la reconfiguración, y también mediante la adición de nuevas funcionalidades.

Al ser un modelo distribuido, se ha generado una relativa independencia entre los niveles del árbol, haciendo que el tráfico de control se reparta de la forma más equitativa posible entre los niveles, para mantener así despejados los recursos del Bootstrap y del

resto de nodos en general.

A nivel de reproducción de vídeo, es interesante destacar que al ser un ALM basado en un único árbol, la complejidad en la gestión de bufferes disminuye considerablemente, al recibir vídeo de una única fuente. Y que gracias a la gestión eficaz que se realiza de las ramas del árbol en cuanto a reconexiones, parece no suponer un problema o una desventaja frente a sistemas con múltiples fuentes y gestión más compleja.

Por último, se ha realizado una implementación tanto del sistema como de un simulador que lo valide, haciendo más fáciles los análisis posteriores y con posibilidades de seguir investigando sobre ello.

5.2. Trabajo Futuro

Aunque en este trabajo se han realizado las bases del sistema que servirán para futuras líneas de investigación, y se puede considerar bastante completo de por sí, quedan en el tintero algunos aspectos que sería interesante tratar. Principalmente habría que definir la gestión de recursos en los nodos, ahora considerados ilimitados, y circunstancias especiales como posibles cambios de nivel o tratamiento de bufferes. También sería interesante centrarse en controlar las salidas desordenadas. La mayoría de estos aspectos ya se han tenido en cuenta para simplificar las labores de trabajo futuras. Además, creemos que es muy interesante seguir investigando en ello debido al gran auge que estos sistemas están teniendo entre los usuarios finales.

Bibliografía

[BBK02] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, page 217. ACM, 2002.

[CDK+03] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, page 313. ACM, 2003.

[CDKR02] M. Castro, P. Druschel, A.M. Kermarrec, and A.I.T. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20(8):1489–1499, 2002.

[CRC+08] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain. Watching television over an IP network. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 71–84. ACM, 2008.

[CRSZ02] Y. Chu, SG Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, 2002.

[HASG07] M. Hosseini, D.T. Ahmed, S. Shirmohammadi, and N.D. Georganas. A survey of application-layer multicast protocols. *IEEE Communications Surveys & Tutorials*, 9(3):58–74, 2007.

[KRAV03] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. *ACM SIGOPS Operating Systems Review*, 37(5):297, 2003.

[LQK+08] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. In *Proc. of IEEE Infocom*. Citeseer, 2008.

[LRLZ08] J. Liu, S.G. Rao, B. Li, and H. Zhang. Opportunities and challenges of peer-to-peer internet video broadcast. *PROCEEDINGS-IEEE*, 96(1):11, 2008.

[NA03] A. Nicolosi and S. Annapureddy. P2PCAST: A peer-to-peer multicast scheme for streaming data. In *1st IRIS Student Workshop (ISW3). Available at: http://www. cs. nyu. edu/nicolosi/P2PCast. ps*. Citeseer, 2003.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 11, pages 329–350. Citeseer, 2001.

[SGG+02] S. Saroiu, P.K. Gummadi, S.D. Gribble, et al. A measurement study of peer-to-peer file sharing systems. In *proceedings of Multimedia Computing and Networking*, volume 2002, page 152. Citeseer, 2002.

[XLKZ07] S. Xie, B. Li, G.Y. Keung, and X. Zhang. Coolstreaming: Design, Theory and Practice. *IEEE Transactions on Multimedia*, 9(8):1661, 2007.

[ZLL05] X. Zhang, J. Liu, and B. Li. On large-scale peer-to-peer live video distribution: Coolstreaming and its preliminary experimental results. In *Proc. MMSP*. Citeseer, 2005.

Apéndice A

Presupuesto

En este capítulo se hará un desglose de las fases de desarrollo del proyecto así como de los costes asociados a la propuesta.

A.1. Tareas

A lo largo del desarrollo de este proyecto se han realizado diferentes tareas hasta poderlo llevar a cabo. Estas tareas quedan reflejadas en la Fig. A.1.

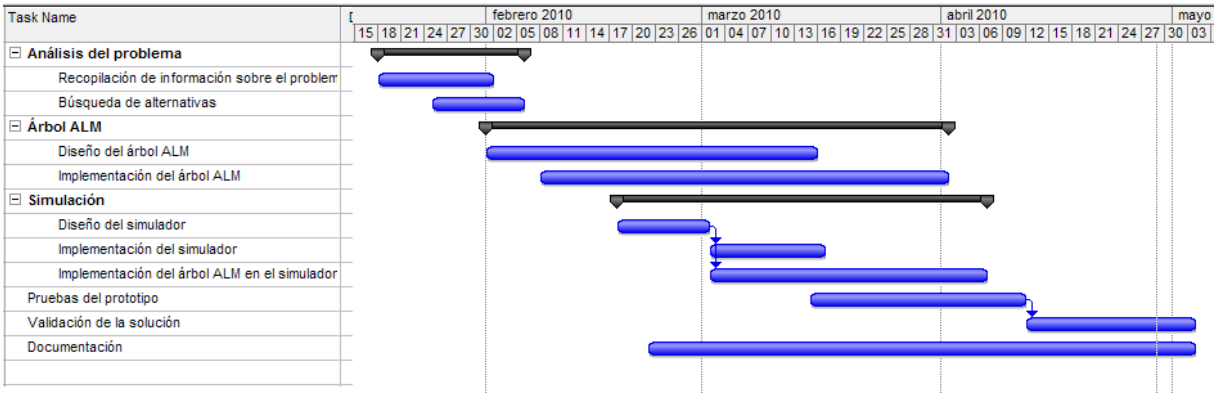

Figura A.1: Diagrama de Gantt con las fases del proyecto

Básicamente, han sido tareas de búsqueda y recopilación de información, diseño, implementación y documentación. El bloque más importante lo constituye el diseño y la implementación.

A.2. Costes

Como se ha podido ver en la Fig. A.1, la duración aproximada del proyecto ha sido de tres meses y medio, y en ese tiempo ha sido necesario cubrir una serie de costes que se detallan a continuación.

A.2.1. Personal

Para la realización del proyecto se ha necesitado cubrir las tareas que realizarían los especialistas que se detallan en la Tabla A.1, junto con el precio de la mano de obra por hora. El jefe de proyecto sería el tutor del mismo.

Tabla A.1: Retribuciones salariales de cada especialista (por hora)

Personal	
Cargo	**Coste (por hora)**
Analista	35,00 €
Arquitecto	40,00 €
Diseñador	40,00 €
Gestión de calidad y pruebas	35,00 €
Jefe de proyecto	45,00 €
Programador	25,00 €

Si se aplican estos costes al número de horas dedicado por cada especialista en función de las fases de diseño y las tareas encomendadas, el resultado se puede ver en la Tabla A.2

A.2.2. Material

En este apartado se detalle el coste de los materiales empleados durante la realización del proyecto. Estos costes se pueden dividir en hardware, software y material fungible.

- **Hardware**

  El único elemento de hardware usado fue un Macbook de 13 pulgadas de aluminio valorado en 1250 € que sufrió un desgaste equivalente a 138,89 €, y que fue sobre el que se realizaron todas las pruebas.

Tabla A.2: Costes de personal

Actividades y recursos							
Actividad		**Personal**					**Totales**
	Jefe de proyecto	**Analista**	**Arquitecto**	**Diseñador**	**Pruebas**	**Programador**	
Análisis del problema	100	30	50	100	0	150	430
Recopilación de información sobre el problema		15					15
Búsqueda de alternativas		15					15
Árbol ALM	10	0	50	100	0	150	310
Diseño del árbol ALM			50	100			150
Implementación del árbol ALM						150	150
Simulación	10	0	75	75	0	100	260
Diseño del simulador				50			50
Implementación del simulador			50				50
Implementación del árbol ALM en el simulador			25	25		100	150
Pruebas del prototipo	10				50	50	110
Validación de la solución	15				50		65
Documentación	15		30	70			115
Horas totales	**160**	**30**	**205**	**345**	**100**	**450**	**1290**
Coste	**7.200,00 €**	**1.050,00 €**	**8.200,00 €**	**13.800,00 €**	**3.500,00 €**	**11.250,00 €**	**45.000,00 €**

- **Software**

  Se refiere a las licencias de software utilizadas. En la Tabla A.3 se puede ver detallado el software utilizado.

Tabla A.3: Costes del software

Software			
Concepto	**Licencias**	**Coste/licencia**	**Coste**
MacOS X Leopard	1	129,00 €	129,00 €
GraphViz	1	0,00 €	0,00 €
Matlab (Licencia universidad)	1	0,00 €	0,00 €
Microsoft Office 2007	1	155,00 €	155,00 €
Latex	1	0,00 €	0,00 €
Eclipse IDE	1	0,00 €	0,00 €
Total			**284,00 €**

- **Material Fungible**

  Por último, en la Tabla A.4 se describen los materiales utilizados durante la realización del proyecto.

Tabla A.4: Costes del material Fungible

Material fungible			
Concepto	**Cantidad**	**Coste unitario**	**Total**
Memorias USB	1	37,00 €	37,00 €
Discos vírgenes (CD / DVD)	2	0,44 €	0,88 €
Disco duro portátil 320 GB	1	97,90 €	97,90 €
Material de oficina	1	25,00 €	25,00 €
Total			**160,78 €**

A.2.3. Transporte

Durante la realización del proyecto se utilizó como medio de transporte un vehículo privado con un coste de 700 €, durante los casi cuatro meses que duró. El consumo medio mensual fue de unos 120 €. El sobrecoste añadido se debe a que hubo que cambiar neumáticos en ese período.

A.2.4. Costes indirectos

En esta Tabla A.5 se resumen el resto de costes derivados del uso de instalaciones durante los tres meses y medio que duró el proyecto.

Tabla A.5: Costes indirectos

Costes indirectos	
Concepto	**Coste**
Alquiler de local	960,00 €
Luz, agua	240,00 €
Servicio de limpieza	240,00 €
Teléfono fijo y red de comunicaciones	280,00 €
Cobertura por bajas *1	1.160,00 €
Seguro a todo riesgo *2	160,00 €
Total	**3.040,00 €**

(*1) En la cobertura por bajas se aplica en caso de tener que contratar personal para cubrir bajas por enfermedad o incapacidad.

(*2) El seguro a todo riesgo incluye cobertura de personal y bienes materiales.

A.2.5. Resumen

Teniendo en cuenta todos los costes relatados anteriormente, el coste total necesario para realizar el proyecto sería de $49{,}323{,}67$ €, tal y como se muestra en la Tabla A.6

Tabla A.6: Resumen de costes

Resumen de costes	
Concepto	**Cantidad**
Mano de obra	45.000,00 €
Equipos	138,89 €
Software	284,00 €
Fungible	160,78 €
Transporte	700,00 €
Costes indirectos	3.040,00 €
Total	**49.323,67 €**

Esta cantidad no tiene en cuenta ni el análisis del riesgo ni los impuestos

A.2.6. Totales

Finalmente, en la Tabla A.7 queda desglosado el balance final de coste del proyecto.

Tabla A.7: Presupuesto total

Presupuesto del PFC	
Concepto	**Cantidad**
Coste total	49.323,67 €
Riesgo (20 %)	9.864,73 €
Beneficio (20 %)	9.864,73 €
Total sin I.V.A.	**69.053,14 €**
I.V.A. (16 %)	11.048,50 €
Total	**80.101,64 €**

Teniendo en cuenta los costes desglosados en los apartados anteriores, el presupuesto total de este proyecto asciende a la cantidad de **OCHENTA MIL CIENTO UNO CON SESENTA Y CUATRO** euros.

Leganés, a 14 de mayo de 2010

El ingeniero proyectista

Fdo. David Díez Hernández